

**ĐẠI HỌC BÁCH KHOA HÀ NỘI  
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

-----□□□□□-----



## **BÁO CÁO BÀI TẬP LỚN**

**LƯU TRỮ VÀ XỬ LÝ DỮ LIỆU LỚN**

**ĐỀ TÀI: PHÂN TÍCH DỮ LIỆU LỚN VỀ  
THỊ TRƯỜNG BẤT ĐỘNG SẢN VIỆT NAM**

**Giáo viên hướng dẫn:** TS. Trần Việt Trung  
TS. Tạ Duy Hoàng

**Mã lớp: 162303 - Nhóm: 26**

**Sinh viên thực hiện:** Phạm Việt Hoàng - 20224854  
Nguyễn Thị Trà My - 20225049  
Đoàn Mạnh Hùng – 20224995  
Phạm Thanh An - 20224911

## MỤC LỤC

<b>CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI.....</b>	<b>5</b>
1. Đặt vấn đề.....	5
1.1. Hiện trạng.....	5
1.2. Vấn đề và giải pháp hiện tại.....	5
2. Tính phù hợp của đề tài với môn học.....	5
3. Mục tiêu và phạm vi của bài tập lớn.....	6
<b>CHƯƠNG 2: NỀN TẢNG LÝ THUYẾT.....</b>	<b>7</b>
1. Thu thập dữ liệu theo lịch trình với Scrapy và Airflow.....	7
1.1. Scrapy – Framework thu thập dữ liệu Web.....	7
1.2. Apache Airflow – Công cụ điều phối và lập lịch workflow.....	7
1.3. Tích hợp Scrapy với Airflow để thu thập dữ liệu theo lịch trình.....	8
2. Kiến trúc và nguyên lý hoạt động của Apache Kafka.....	8
2.1. Mô hình lưu trữ Log-Structured.....	8
2.2. Mô hình Pub-Sub với Topic và Partition.....	8
2.3. Cấu trúc Cluster và các thành phần chính.....	9
2.4. Cơ chế phân phối và sao lưu dữ liệu.....	9
3. Xử lý dữ liệu lớn với Apache Spark.....	9
3.1. Giới thiệu về Spark Streaming.....	10
3.2. Mô hình xử lý Micro-Batch.....	10
3.3. Kiến trúc xử lý dựa trên DStream.....	10
3.4. Khả năng mở rộng và phục hồi lỗi.....	11
<b>CHƯƠNG 3: THIẾT KẾ KIẾN TRÚC.....</b>	<b>12</b>
1. Tổng quan hệ thống.....	12
2. Chi tiết các thành phần và vai trò đối với hệ thống.....	13
2.1. Kubernetes (K8s).....	13
2.2. Scrapy.....	14
2.3. Airflow.....	15
2.4. Kafka.....	16
2.5. Spark.....	17
2.6. MinIO.....	18
2.7. ElasticSearch.....	19
2.8. Hệ thống đa tác tử (multi-agent).....	19
2.9. Streamlit.....	21
3. Luồng dữ liệu và tương tác giữa các thành phần hệ thống.....	22
<b>CHƯƠNG 4: TRIỂN KHAI HỆ THỐNG.....</b>	<b>24</b>
1. Môi trường và chiến lược triển khai hệ thống.....	24
1.1. Môi trường triển khai phần cứng và hệ điều hành.....	24
1.2. Chuẩn bị môi trường triển khai.....	25
1.3. Chiến lược triển khai hạ tầng.....	25
2. Triển khai hạ tầng Kubernetes.....	25

3. Triển khai thành phần thu thập và tiền xử lý dữ liệu.....	27
3.1. Dịch vụ thu thập dữ liệu.....	27
3.2. Tiền xử lý dữ liệu.....	28
3.3. API thu thập dữ liệu.....	29
3.4. Dịch vụ lập lịch thu thập dữ liệu.....	29
4. Triển khai thành phần xử lý và lưu trữ dữ liệu.....	29
4.1. Lưu trữ dữ liệu.....	30
4.2. Xử lý dữ liệu.....	34
5. Triển khai thành phần đa tác tử (Multi-Agent).....	38
5.1. Tổng quan triển khai.....	38
5.2. Hệ thống Đa tác tử.....	38
5.3. Các hệ thống hỗ trợ.....	39
6. Triển khai thành phần trực quan hóa và giao diện Chatbot.....	42
6.1. Giao diện Xác thực người dùng.....	42
6.2. Giao diện Chatbot.....	43
6.3. Thành phần trực quan hóa dữ liệu.....	44
7. Source code và documentation.....	48
<b>CHƯƠNG 5: TỔNG KẾT VÀ BÀI HỌC KINH NGHIỆM.....</b>	<b>49</b>
1. Bài học về Thu thập dữ liệu (Data Ingestion).....	49
1.1. Kinh nghiệm về đồng bộ hóa dữ liệu đa nguồn.....	49
1.2. Kinh nghiệm về khử trùng lặp nội dung bằng NLP.....	50
2. Bài học về Xử lý luồng (Stream Processing).....	50
2.1. Kinh nghiệm về đảm bảo tính toàn vẹn dữ liệu.....	50
3. Bài học về Lưu trữ dữ liệu.....	51
3.1. Kinh nghiệm về tối ưu hóa định dạng lưu trữ.....	51
3.2. Kinh nghiệm về phân vùng dữ liệu.....	52
4. Bài học về tích hợp hệ thống.....	52
4.1. Kinh nghiệm phát hiện dịch vụ trong môi trường động.....	52
5. Bài học về Tối ưu hiệu năng.....	53
5.1. Kinh nghiệm Caching trong tính toán lặp.....	53
6. Bài học về mở rộng hệ thống.....	54
6.1. Kinh nghiệm về Hybrid Scaling.....	54
7. Bài học về bảo mật và quản trị hệ thống.....	54
7.1. Kinh nghiệm về quản lý truy cập.....	54
8. Bài học về khả năng chịu lỗi.....	55
8.1. Kinh nghiệm thiết kế hệ thống với tư duy "Crash-first".....	55
9. Bài học về hệ thống Backend & Multi-agent.....	56
9.1. Kinh nghiệm về quản lý ngữ cảnh và tối ưu chi phí Token.....	56
9.2. Kinh nghiệm về kiểm soát áô giác của Agent.....	56
9.3. Kinh nghiệm Debugging trong hệ thống bất định.....	57
10. Bài học về Trực quan hóa dữ liệu.....	58
10.1. Kinh nghiệm về xử lý phân phối dữ liệu lệch trong Heatmap.....	58

**DANH MỤC HÌNH ẢNH**

Hình 1. Kiến trúc tổng quan toàn bộ hệ thống.....	12
Hình 2. Kiến trúc chi tiết hệ thống đa tác tử.....	20
Hình 3. Các instance được triển khai trên EC2.....	25
Hình 4. Tạo vai trò IAM với quyền truy cập dịch vụ EBS.....	26
Hình 5. Airflow lập lịch chạy crawler.....	29
Hình 6. Các broker trong cụm Kafka.....	30
Hình 7. Luồng giao tiếp của crawler local và remote kafka.....	31
Hình 8. Giao diện MinIO quản lý dữ liệu trong bucket.....	32
Hình 9. Giao diện hệ thống Elasticsearch.....	33
Hình 10. Giao diện Kibana giúp người dùng theo dõi Elasticsearch hiệu quả hơn.....	34
Hình 11. Log của quá trình xử lý dữ liệu luồng từ Kafka.....	37
Hình 12. Log của quá trình xử lý lô dữ liệu.....	38
Hình 13. Triển khai hệ thống đa tác tử.....	38
Hình 14. Tương tác giữa các tác tử trong hệ thống.....	39
Hình 15. Giao diện hệ thống MongoDB.....	40
Hình 16. Giao diện hệ thống Zep lưu trữ bộ nhớ.....	41
Hình 17. Các API trên giao diện FastAPI.....	42
Hình 18. Giao diện hệ thống Đăng ký.....	43
Hình 19. Giao diện hệ thống Đăng nhập.....	43
Hình 20. Giao diện hệ thống Chatbot UI.....	44
Hình 21. Chatbot phân tích và đưa ra lời khuyên rõ ràng.....	44
Hình 22. Giao diện hệ thống trực quan hóa.....	44
Hình 23. Biểu đồ cột thống kê giá và diện tích các loại hình bất động sản tại các quận huyện ở từng tỉnh/thành chọn trước.....	45
Hình 24. Bảng số liệu thể hiện Giá Trung Bình, Đơn giá/m <sup>2</sup> và Diện Tích của từng các quận huyện ở từng tỉnh thành.....	46
Hình 25. Biểu đồ đường thể hiện biến động giá theo từng khoảng thời gian, tại từng khu vực chọn trước.....	46
Hình 26. Bảng số liệu thể hiện Giá Trung Bình và Giá/m <sup>2</sup> của từng địa phương trong 1 khoảng thời gian nhất định.....	47
Hình 27. Bản đồ nhiệt thể hiện mật độ giá trung bình của từng địa phương.....	47

## LỜI NÓI ĐẦU

Trong kỷ nguyên số hiện nay, dữ liệu đã và đang trở thành "tài sản" chiến lược, đóng vai trò then chốt trong việc thúc đẩy sự phát triển của mọi lĩnh vực từ kinh tế, xã hội đến công nghệ. Sự bùng nổ của Internet, các nền tảng mạng xã hội và hệ thống giao dịch trực tuyến đã tạo ra một lượng dữ liệu khổng lồ với đặc trưng 5V. Tuy nhiên, thách thức đặt ra là làm thế nào để thu thập, lưu trữ và khai thác nguồn tài nguyên này một cách hiệu quả để biến những con số thành thông tin có giá trị cho quá trình ra quyết định.

Tại Việt Nam, thị trường bất động sản là một trong những lĩnh vực có tốc độ biến động nhanh và dữ liệu cực kỳ phân tán. Với hàng nghìn tin đăng mỗi ngày trên các nền tảng khác nhau, người dùng và các nhà đầu tư thường xuyên đối mặt với tình trạng nhiễu thông tin, thiếu tính minh bạch và sự chênh lệch về giá cả. Việc ứng dụng các công nghệ dữ liệu lớn để xây dựng một hệ thống xử lý dòng dữ liệu tập trung, làm sạch và phân tích xu hướng thị trường không chỉ là một bài toán công nghệ thú vị mà còn mang lại giá trị thực tiễn cao cho cộng đồng.

Nhận thức được tầm quan trọng đó, nhóm chúng em đã quyết định thực hiện đề tài: “Xây dựng hệ thống phân tích dữ liệu lớn về thị trường bất động sản Việt Nam”. Bài tập lớn này tập trung vào việc nghiên cứu và ứng dụng các nền tảng công nghệ hiện đại trong hệ sinh thái Big Data như Apache Kafka để xử lý dòng sự kiện, Apache Spark để tính toán phân tán, cùng các giải pháp lưu trữ trên nền tảng điện toán đám mây. Thông qua quá trình thực hiện, nhóm hy vọng có thể làm rõ quy trình xử lý dữ liệu từ giai đoạn thu thập thô đến khi trực quan hóa, đồng thời giải quyết các bài toán về tối ưu hóa lưu trữ và xử lý dữ liệu phi cấu trúc trong thực tế.

Mặc dù đã có nhiều cố gắng trong quá trình nghiên cứu và thực hiện, nhưng do giới hạn về kiến thức và thời gian, báo cáo chắc chắn không tránh khỏi những thiếu sót. Nhóm chúng em rất mong nhận được sự đóng góp ý kiến từ thầy và các bạn để đề tài được hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

## CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

### 1. Đặt vấn đề

#### 1.1. Hiện trạng

Trong thời đại công nghệ số và trí tuệ nhân tạo phát triển mạnh mẽ, dữ liệu đã trở thành một trong những nguồn tài nguyên quan trọng nhất, đóng vai trò then chốt trong việc hỗ trợ ra quyết định và tối ưu hóa hoạt động kinh doanh. Đặc biệt, sự bùng nổ của dữ liệu phi cấu trúc từ các nền tảng trực tuyến, mạng xã hội và các hệ thống giao dịch điện tử đã đặt ra yêu cầu cấp thiết về khả năng thu thập, lưu trữ và phân tích dữ liệu lớn một cách hiệu quả.

Đất nước ta hiện nay đang chứng kiến sự phát triển mạnh mẽ của thị trường bất động sản với số lượng tin đăng, giao dịch mua bán và cho thuê gia tăng nhanh chóng mỗi ngày. Các thông tin bất động sản hiện nay chủ yếu được đăng tải trên nhiều website, sàn giao dịch trực tuyến và mạng xã hội khác nhau, tạo ra một lượng lớn dữ liệu đa dạng về giá cả, diện tích, vị trí, loại hình nhà đất và các đặc điểm liên quan.

#### 1.2. Vấn đề và giải pháp hiện tại

Thực tế cho thấy dữ liệu bất động sản hiện vẫn còn tồn tại nhiều hạn chế như:

- Dữ liệu bất động sản đến từ nhiều nguồn khác nhau (website rao vặt, sàn giao dịch, mạng xã hội), chủ yếu ở dạng bán cấu trúc và phi cấu trúc.
- Thông tin bị trùng lặp, thiếu chuẩn hóa, nhiều bài đăng không đầy đủ hoặc chứa dữ liệu nhiễu.
- Người dùng gặp khó khăn trong việc tổng hợp, phân tích xu hướng giá, khu vực tiềm năng hoặc so sánh các bất động sản tương tự.
- Chưa có hệ thống tập trung để xử lý khối lượng dữ liệu lớn một cách tự động và hiệu quả.

Điều này gây khó khăn cho người dùng trong việc tìm kiếm, so sánh và đánh giá các cơ hội đầu tư hoặc nhu cầu nhà ở phù hợp. Do đó, việc xây dựng một hệ thống thu thập, xử lý và phân tích dữ liệu bất động sản dựa trên các công nghệ Big Data là một bài toán có tính thực tiễn và giá trị cao. Trong bài tập lớn này, nhóm đề xuất xây dựng một data pipeline tự động để thu thập thông tin bất động sản từ các nền tảng trực tuyến. Hệ thống sẽ sử dụng các công nghệ hiện đại như Airflow để điều phối, lưu trữ trên Data Lake và sử dụng Apache Spark để xử lý dữ liệu lớn phi cấu trúc thành dữ liệu có cấu trúc. Cuối cùng, dữ liệu được phân tích và trực quan hóa để cung cấp cái nhìn toàn cảnh về xu hướng giá cả và chất lượng thị trường.

### 2. Tính phù hợp của đề tài với môn học

Đề tài phân tích dữ liệu bất động sản đáp ứng đầy đủ các đặc trưng 5V của Big Data:

- Khối lượng dữ liệu lớn (Volume): Số lượng tin đăng bất động sản phát sinh mỗi ngày rất lớn, bao gồm nhiều thuộc tính như giá, diện tích, vị trí, mô tả chi tiết và hình ảnh, tạo ra khối lượng dữ liệu đáng kể cần được xử lý.
- Tốc độ phát sinh dữ liệu (Velocity): Dữ liệu liên tục được cập nhật theo thời gian thực khi người dùng đăng tin mới, chỉnh sửa hoặc gỡ bỏ bài đăng, đòi hỏi hệ thống có khả năng xử lý dữ liệu nhanh và linh hoạt.

- Tính đa dạng của dữ liệu (Variety): Dữ liệu tồn tại dưới nhiều dạng khác nhau như dữ liệu bảng (giá, diện tích), văn bản (mô tả bất động sản), và dữ liệu bán cấu trúc từ các website, đáp ứng đặc trưng đa dạng của Big Data.
- Giá trị dữ liệu (Value): Việc phân tích dữ liệu bất động sản giúp người mua, người bán và nhà đầu tư đưa ra quyết định chính xác hơn, đồng thời hỗ trợ đánh giá xu hướng thị trường và tiềm năng phát triển của từng khu vực.
- Tính xác thực của dữ liệu (Veracity): Dữ liệu thu thập từ các sàn giao dịch uy tín kết hợp với quá trình làm sạch, loại bỏ trùng lặp và dữ liệu nhiễu giúp nâng cao độ tin cậy của tập dữ liệu cuối cùng.

### 3. Mục tiêu và phạm vi của bài tập lớn

#### Mục tiêu:

- Xây dựng hệ thống tự động thu thập dữ liệu từ các trang bất động sản lớn ở Việt Nam.
- Thiết kế 1 pipeline sử dụng Kiến trúc Lambda và K8s để điều phối, từ quá trình thu thập cho tới xử lý, lưu trữ và trực quan hóa dữ liệu.
- Sử dụng đầy đủ các công cụ mạnh như Airflow, Scrapy, Kafka, Speaker, Elasticsearch, MinIO trong quá trình triển khai hệ thống
- Trực quan hóa dữ liệu qua Streamlit, tận dụng nguồn data có sẵn xây dựng hệ thống Multi-agent chatbot về bất động sản.

#### Phạm vi và giới hạn:

- Đối tượng: Các bài đăng bán và cho thuê bất động sản nhà riêng, chung cư, biệt thự, nhà mặt phố, đất đai...
- Khu vực: Các quận, huyện thuộc 63 tỉnh thành tại Việt Nam.
- Thời gian dữ liệu: Thu thập và phân tích dữ liệu trong giai đoạn cuối năm 2025.

## CHƯƠNG 2: NỀN TẢNG LÝ THUYẾT

### 1. Thu thập dữ liệu theo lịch trình với Scrapy và Airflow

#### 1.1. Scrapy – Framework thu thập dữ liệu Web

Scrapy là một framework mã nguồn mở viết bằng Python, được thiết kế chuyên biệt cho việc thu thập dữ liệu web (web crawling và web scraping) với hiệu năng cao. Scrapy cho phép tự động gửi các yêu cầu HTTP, phân tích nội dung HTML và trích xuất dữ liệu theo các quy tắc được định nghĩa trước.

Các thành phần chính trong Scrapy bao gồm:

- Spider: định nghĩa logic thu thập dữ liệu, bao gồm các URL bắt đầu, cách di chuyển giữa các trang và quy tắc trích xuất dữ liệu.
- Item: mô hình dữ liệu dùng để lưu trữ các trường thông tin thu thập được.
- Item Pipeline: xử lý dữ liệu sau khi thu thập, ví dụ làm sạch dữ liệu, loại bỏ trùng lặp và lưu trữ vào cơ sở dữ liệu hoặc file.
- Downloader & Scheduler: quản lý việc gửi request và nhận response một cách bất đồng bộ.

Nhờ kiến trúc bất đồng bộ và khả năng xử lý song song, Scrapy có thể thu thập một lượng lớn dữ liệu trong thời gian ngắn.

#### 1.2. Apache Airflow – Công cụ điều phối và lập lịch workflow

Trong thực tế, dữ liệu trên các website không cố định mà liên tục được cập nhật, thêm mới hoặc chỉnh sửa. Vì vậy, việc thu thập dữ liệu cần được thực hiện theo chu kỳ thời gian định trước, chẳng hạn theo giờ, theo ngày hoặc theo tuần. Thu thập dữ liệu theo lịch trình giúp:

- Đảm bảo dữ liệu luôn được cập nhật mới nhất.
- Giảm thiểu việc thu thập dữ liệu trùng lặp.
- Hỗ trợ phân tích xu hướng theo thời gian.
- Tự động hóa toàn bộ quy trình ingestion dữ liệu.

Tuy nhiên, Scrapy không được thiết kế để quản lý lịch trình và giám sát workflow phức tạp. Do đó, cần kết hợp với một công cụ điều phối quy trình như Apache Airflow.

Airflow là một nền tảng điều phối workflow mã nguồn mở, cho phép xây dựng, lập lịch và giám sát các quy trình xử lý dữ liệu dưới dạng Directed Acyclic Graph (DAG). Mỗi DAG biểu diễn một pipeline xử lý dữ liệu bao gồm nhiều task có thứ tự và mối quan hệ phụ thuộc rõ ràng.

Các thành phần chính của Airflow bao gồm:

- DAG: mô tả toàn bộ workflow xử lý dữ liệu.
- Task / Operator: đại diện cho các bước xử lý cụ thể, ví dụ chạy spider Scrapy, lưu dữ liệu hoặc kiểm tra kết quả.
- Scheduler: chịu trách nhiệm kích hoạt các DAG theo lịch trình đã định nghĩa.
- Executor: thực thi các task song song trên một hoặc nhiều node.
- Web UI: giao diện trực quan để theo dõi trạng thái, log và lịch sử chạy pipeline.

Airflow cho phép cấu hình lịch chạy linh hoạt (cron-based), hỗ trợ retry, cảnh báo lỗi và đảm bảo tính ổn định cho pipeline thu thập dữ liệu.

### 1.3. Tích hợp Scrapy với Airflow để thu thập dữ liệu theo lịch trình

Trong hệ thống Big Data, Scrapy và Airflow thường được kết hợp để xây dựng pipeline thu thập dữ liệu tự động, trong đó:

- Scrapy đảm nhiệm vai trò thu thập và trích xuất dữ liệu từ các nguồn web.
- Airflow đảm nhiệm vai trò điều phối, lập lịch và giám sát toàn bộ quá trình thu thập dữ liệu.

Một workflow điển hình bao gồm:

- Airflow Scheduler kích hoạt DAG theo lịch định sẵn.
- Task đầu tiên thực thi spider Scrapy để crawl dữ liệu.
- Dữ liệu sau khi thu thập được lưu vào hệ thống lưu trữ trung gian (file, object storage hoặc message queue).
- Các task tiếp theo kiểm tra dữ liệu, xử lý lỗi và chuyển dữ liệu sang các bước xử lý tiếp theo trong pipeline Big Data.

## 2. Kiến trúc và nguyên lý hoạt động của Apache Kafka

Apache Kafka là một nền tảng xử lý dòng sự kiện phân tán (Distributed Event Streaming Platform) được phát triển bởi Apache Software Foundation. Kafka được thiết kế nhằm giải quyết bài toán thu thập, lưu trữ và xử lý dữ liệu luồng (streaming data) với khối lượng lớn, tốc độ cao và yêu cầu độ trễ thấp – những đặc trưng điển hình của hệ thống Big Data hiện đại.

Kafka cho phép truyền tải dữ liệu theo thời gian thực giữa các hệ thống khác nhau một cách tin cậy, đồng thời hỗ trợ mở rộng quy mô theo chiều ngang (horizontal scaling). Nhờ đó, Kafka được ứng dụng rộng rãi trong các bài toán như xử lý log, thu thập dữ liệu sự kiện, phân tích hành vi người dùng và xử lý dữ liệu streaming.

### 2.1. Mô hình lưu trữ Log-Structured

Kafka sử dụng mô hình lưu trữ dữ liệu dạng log-structured, trong đó dữ liệu được tổ chức thành các dòng thông điệp (message log) và được ghi tuần tự xuống đĩa. Mỗi thông điệp bao gồm một tập byte dữ liệu và được ghi vào các phân vùng (Partition) theo thứ tự thời gian.

Đặc điểm quan trọng của mô hình này là:

- Dữ liệu chỉ được ghi nối tiếp (append-only), không hỗ trợ chỉnh sửa hoặc xóa trực tiếp.
- Việc ghi tuần tự giúp Kafka tận dụng tối đa hiệu năng I/O của đĩa, từ đó đạt được thông lượng cao.
- Dữ liệu được giữ lại trong một khoảng thời gian xác định hoặc theo dung lượng, phục vụ cho việc đọc lại và xử lý lại dữ liệu khi cần thiết.

Mô hình log-structured giúp Kafka xử lý hiệu quả các luồng dữ liệu lớn và liên tục, đáp ứng yêu cầu về Volume và Velocity trong Big Data.

### 2.2. Mô hình Pub-Sub với Topic và Partition

Kafka hoạt động theo mô hình Publisher – Subscriber (Pub-Sub), trong đó các thành phần chính bao gồm:

- Producer: là các ứng dụng hoặc hệ thống gửi dữ liệu (thông điệp) vào Kafka.
- Consumer: là các ứng dụng đăng ký nhận dữ liệu từ Kafka để xử lý hoặc lưu trữ.

Dữ liệu trong Kafka được phân loại theo Topic, mỗi Topic đại diện cho một luồng dữ liệu logic. Để hỗ trợ xử lý song song và phân tán, mỗi Topic được chia thành nhiều Partition.

Trong mỗi Partition:

- Các thông điệp được sắp xếp theo thứ tự.
- Mỗi thông điệp được gán một chỉ số duy nhất gọi là Offset, giúp consumer xác định vị trí đọc dữ liệu.

Cơ chế Partition cho phép nhiều consumer có thể đọc dữ liệu song song, từ đó nâng cao hiệu năng xử lý và khả năng mở rộng của hệ thống – một đặc trưng quan trọng trong các ứng dụng Big Data.

### 2.3. Cấu trúc Cluster và các thành phần chính

Kafka được triển khai dưới dạng một Kafka Cluster, bao gồm nhiều máy chủ hoạt động phối hợp với nhau. Các thành phần chính trong hệ thống Kafka bao gồm:

- Broker: là các máy chủ chịu trách nhiệm lưu trữ dữ liệu và xử lý các yêu cầu đọc/ghi từ producer và consumer. Mỗi broker có thể chứa nhiều partition thuộc các topic khác nhau.
- ZooKeeper: được sử dụng để quản lý metadata của cluster, điều phối các broker, theo dõi trạng thái leader của các partition và hỗ trợ quá trình bầu chọn leader khi xảy ra sự cố.
- Topic: là đơn vị logic dùng để phân loại và quản lý các luồng dữ liệu.
- Producer / Consumer: các thành phần tương tác trực tiếp với Kafka để gửi và nhận dữ liệu.

Nhờ kiến trúc phân tán, Kafka cho phép mở rộng hệ thống dễ dàng bằng cách thêm broker mới vào cluster mà không làm gián đoạn hoạt động của toàn hệ thống.

### 2.4. Cơ chế phân phối và sao lưu dữ liệu

Để đảm bảo độ tin cậy và khả năng chịu lỗi, Kafka hỗ trợ cơ chế Replication. Theo đó, mỗi Partition có thể được sao chép thành nhiều bản trên các broker khác nhau. Trong số các bản sao:

- Một broker giữ vai trò Leader, chịu trách nhiệm xử lý tất cả các thao tác đọc và ghi.
- Các broker còn lại là Follower, có nhiệm vụ đồng bộ dữ liệu từ Leader.

Khi broker giữ vai trò Leader gặp sự cố, Kafka sẽ tự động bầu chọn một Follower khác làm Leader mới, đảm bảo hệ thống vẫn tiếp tục hoạt động mà không làm mất dữ liệu. Cơ chế này giúp Kafka đáp ứng yêu cầu tính sẵn sàng cao trong các hệ thống Big Data.

## 3. Xử lý dữ liệu lớn với Apache Spark

Apache Spark là một nền tảng xử lý dữ liệu phân tán được thiết kế để xử lý và phân tích dữ liệu lớn với hiệu năng cao. Spark cho phép xử lý dữ liệu trên bộ nhớ (in-memory processing), giúp giảm đáng kể thời gian đọc/ghi dữ liệu từ đĩa so với các hệ thống xử lý batch truyền thống như Hadoop MapReduce.

Một trong những đặc điểm quan trọng của Spark là cơ chế lazy evaluation, theo đó các phép biến đổi dữ liệu chỉ được ghi nhận và chỉ thực sự thực thi khi có yêu cầu trả

kết quả. Cơ chế này giúp Spark tối ưu hóa kế hoạch thực thi, từ đó nâng cao hiệu năng và khả năng mở rộng.

Spark hỗ trợ nhiều ngôn ngữ lập trình phổ biến như Scala, Java, Python và R, giúp người dùng dễ dàng tích hợp Spark vào các hệ thống hiện có và triển khai các ứng dụng xử lý dữ liệu phức tạp trong môi trường Big Data.

### 3.1. Giới thiệu về Spark Streaming

Spark Streaming là một thành phần mở rộng trong hệ sinh thái Apache Spark, được thiết kế để xử lý dữ liệu dòng (streaming data) - tức các luồng dữ liệu liên tục được sinh ra theo thời gian thực từ nhiều nguồn khác nhau như cảm biến IoT, mạng xã hội, hệ thống giao dịch điện tử hoặc log hệ thống.

Khác với mô hình xử lý theo lô truyền thống (batch processing), Spark Streaming cho phép xử lý dữ liệu gần thời gian thực (near real-time) bằng cách tận dụng sức mạnh xử lý phân tán và in-memory của Spark Core. Nhờ đó, Spark Streaming phù hợp với các bài toán Big Data yêu cầu vừa xử lý khối lượng lớn dữ liệu, vừa đảm bảo độ trễ thấp.

### 3.2. Mô hình xử lý Micro-Batch

Spark Streaming áp dụng mô hình micro-batch, trong đó luồng dữ liệu đầu vào được chia thành các lô nhỏ (mini-batches) theo chu kỳ thời gian cố định, ví dụ mỗi 1 giây hoặc 5 giây. Mỗi micro-batch sau đó được xử lý như một batch thông thường bằng Spark Engine.

Quy trình xử lý dữ liệu trong Spark Streaming bao gồm các bước chính:

- Thu thập dữ liệu đầu vào từ các nguồn như Apache Kafka, Apache Flume hoặc socket TCP.
- Chia dòng dữ liệu thành các micro-batch thông qua cơ chế Receiver hoặc Direct Stream.
- Thực hiện các phép biến đổi dữ liệu tương tự Spark Core như map, filter, flatMap, reduceByKey,...
- Quản lý trạng thái khi cần, chẳng hạn như đếm số lượng sự kiện theo người dùng hoặc theo khoảng thời gian.
- Đẩy dữ liệu đầu ra đến các hệ thống lưu trữ hoặc xử lý tiếp theo như HDFS, Elasticsearch, cơ sở dữ liệu quan hệ hoặc NoSQL.

Mô hình micro-batch giúp Spark Streaming đạt được sự cân bằng giữa hiệu năng, khả năng mở rộng và tính ổn định trong xử lý dữ liệu dòng.

### 3.3. Kiến trúc xử lý dựa trên DStream

Spark Streaming xây dựng luồng xử lý dữ liệu dựa trên cấu trúc DStream (Discretized Stream). Một DStream đại diện cho một chuỗi các RDD (Resilient Distributed Dataset) được tạo ra theo từng khoảng thời gian.

Mỗi RDD trong DStream tương ứng với dữ liệu thu thập được trong một micro-batch cụ thể. Nhờ việc kế thừa các đặc tính của RDD, DStream có khả năng:

- Phân tán dữ liệu trên nhiều node trong cluster.
- Thực hiện các phép biến đổi song song.
- Đảm bảo khả năng phục hồi khi xảy ra lỗi.

### **3.4. Khả năng mở rộng và phục hồi lỗi**

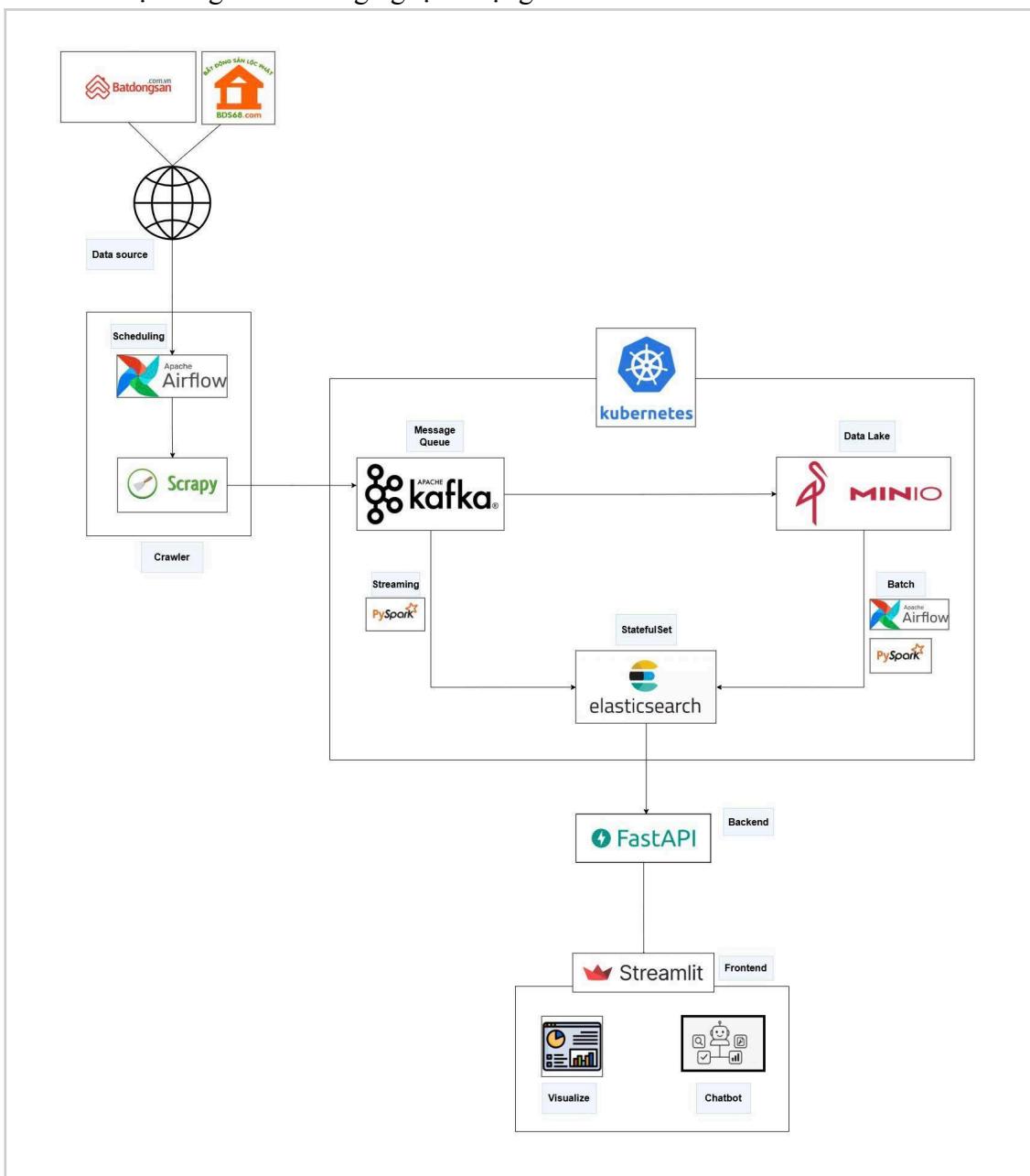
Spark Streaming hỗ trợ mở rộng theo chiều ngang (scale-out) nhờ kiến trúc phân tán của Spark. Việc tăng năng lực xử lý có thể thực hiện đơn giản bằng cách bổ sung thêm các node vào cluster Spark.

Bên cạnh đó, hệ thống có khả năng chịu lỗi thông qua cơ chế lineage của RDD. Spark ghi lại chuỗi các phép biến đổi đã áp dụng lên dữ liệu, cho phép tự động tính toán lại các RDD bị mất khi xảy ra sự cố, đảm bảo tính toàn vẹn của quá trình xử lý dữ liệu.

## CHƯƠNG 3: THIẾT KẾ KIẾN TRÚC

## 1. Tổng quan hệ thống

Kiến trúc hệ thống và các công nghệ sử dụng:



### **Hình 1. Kiến trúc tổng quan toàn bộ hệ thống**

Nhóm lựa chọn triển khai hệ thống lưu trữ và xử lý dữ liệu lớn theo kiến trúc Lambda. Kiến trúc Lambda là một mô hình thiết kế để vừa xử lý được khối lượng dữ liệu lịch sử khổng lồ (Batch), vừa xử lý được dữ liệu mới sinh ra tức thì (Streaming) với độ trễ thấp, nhờ đó đạt được sự cân bằng giữa tốc độ và độ chính xác.

Kiến trúc Lambda gồm 3 tầng:

- **Tầng xử lý theo lô (Batch Layer):** Batch Layer có một kho lưu trữ dữ liệu gốc, bất biến, không bao giờ bị sửa đổi. Mặt khác cũng có cài đặt tính toán lại thông tin dựa trên toàn bộ lịch sử nhằm cung cấp batch view có độ chính xác cao.

Tầng xử lý theo lô đánh đổi tốc độ phục vụ với độ chính xác, chấp nhận độ trễ lớn, do đó không thể chạy liên tục mà chỉ chạy định kì. Trong dự án, nhóm triển khai lưu trữ dữ liệu bằng MinIO, xử lý lô lịch sử dữ liệu bằng PySpark trong hệ sinh thái Apache Spark, lập lịch chạy định kì bằng Airflow.

- **Tầng xử lý theo luồng (Stream Layer):** Stream Layer chịu trách nhiệm xử lý dữ liệu luồng thời gian thực, ưu tiên tốc độ đáp ứng dịch vụ, chấp nhận đánh đổi độ chính xác so với Batch Layer. Logic xử lý trong Stream Layer cũng đơn giản hơn, chỉ sử dụng các dữ liệu được chuyển đến trong 1 thời điểm, đảm bảo độ trễ thấp. Trong dự án, nhóm triển khai một hàng đợi thông điệp (message queue) Kafka để lưu trữ tạm thời các thông tin truyền đến, tránh để thất thoát thông tin khi consumer chưa xử lý kịp; triển khai xử lý luồng dữ liệu bằng PySpark.
- **Tầng tích hợp dữ liệu (Serving Layer):** Serving Layer là nơi tổng hợp thông tin từ tầng xử lý theo lô và tầng xử lý theo luồng để cung cấp cho người dùng cuối một góc nhìn tổng quát vừa đáp ứng được tốc độ phục vụ và độ chính xác của thông tin. Trong dự án, nhóm sử dụng Elasticsearch để triển khai.

Toàn bộ kiến trúc Lambda được triển khai trên hạ tầng Kubernetes (hay K8s) nhằm đảm bảo tính ổn định, tin cậy, sẵn sàng.

Bộ thu thập dữ liệu được nhóm triển khai bằng Scrapy, thực hiện thu thập trên các trang web bất động sản đã chọn. Và quá trình thu thập được thực hiện định kì một cách tự động bằng Airflow.

Ngoài ra, nhóm triển khai thêm các thành phần phụ: trực quan hóa dữ liệu từ Elasticsearch và tác tử hỏi đáp thông tin bất động sản để mô phỏng vai trò của người dùng khi tương tác với đầu ra cuối cùng được cung cấp bởi tầng tích hợp dữ liệu của hệ thống.

## 2. Chi tiết các thành phần và vai trò đối với hệ thống

### 2.1. Kubernetes (K8s)

Kubernetes (K8s) đóng vai trò là hệ thống điều phối container cho toàn bộ project. Với tính chất phức tạp của hệ thống bao gồm nhiều dịch vụ phân tán (Kafka Cluster, Spark Workers, Airflow, Web Scraper), Kubernetes chịu trách nhiệm:

- Quản lý vòng đời ứng dụng: Tự động triển khai, khởi động, dừng và khởi động lại các container của các dịch vụ như Kafka hay Spark.
- Điều phối tài nguyên: Phân bổ CPU và RAM hợp lý cho từng thành phần. Ví dụ: Cấp nhiều tài nguyên hơn cho Spark khi cần xử lý lô dữ liệu lớn và giảm tài nguyên khi hệ thống nhàn rỗi.
- Quản lý mạng: Đảm bảo các thành phần giao tiếp được với nhau ổn định thông qua cơ chế Service và DNS nội bộ mà không cần cấu hình IP tĩnh thủ công
- Kubernetes cung cấp các abstraction (lớp trừu tượng) mạnh mẽ giúp vận hành hệ thống Big Data ổn định.

#### Lợi ích:

- High Availability: Nếu một node hoặc pod bị chết, K8s tự động phát hiện và tạo lại pod mới trên node khác.
- Scalability: Dễ dàng mở rộng cụm Kafka hoặc Spark Cluster chỉ bằng một câu lệnh đơn giản hoặc cấu hình tự động.

- Môi trường nhất quán: Đảm bảo môi trường chạy trên máy local giống hệt với môi trường production trên Cloud.

#### Nhược điểm:

- Độ phức tạp cao: Việc cài đặt, cấu hình và vận hành một cluster K8s đòi hỏi kiến thức sâu về hệ thống và mạng.
- Tốn tài nguyên: K8s cần một lượng tài nguyên nhất định (CPU/RAM) cho các thành phần quản lý trước khi chạy ứng dụng thực tế.

So với Docker Compose thường chỉ chạy trên một máy chủ vật lý (Single Node). Nếu khói lượng xử lý Spark hoặc Kafka vượt quá giới hạn phần cứng của máy đó, hệ thống sẽ bị treo. Việc mở rộng thủ công rất khó khăn. Kubernetes được thiết kế để chạy trên một cụm nhiều máy chủ (Multi-node Cluster). K8s cho phép nhóm dễ dàng thêm node mới vào cluster để tăng sức mạnh tính toán mà không làm gián đoạn hệ thống. Trong quá trình phát triển, nhóm thường xuyên cập nhật mã nguồn Scrapy hoặc Spark, Kubernetes hỗ trợ Rolling Updates, cho phép cập nhật từng Pod một cách tuần tự. Hệ thống vẫn phục vụ bình thường trong quá trình cập nhật phiên bản mới. Việc sử dụng K8s giúp nhóm tận dụng tối đa hạ tầng của AWS.

#### 2.2. Scrapy

Scrapy là một framework mã nguồn mở mạnh mẽ được viết bằng Python, chuyên dụng cho việc trích xuất dữ liệu từ website. Trong hệ thống phân tích bất động sản, Scrapy đóng vai trò là Data Collector với các nhiệm vụ cụ thể:

- Duyệt trang (Crawling): Tự động điều hướng qua hàng nghìn trang danh mục tin đăng tại Hà Nội trên các website như [bds.com.vn](http://bds.com.vn), [bds68.com.vn](http://bds68.com.vn), ...
- Trích xuất dữ liệu (Extraction): Sử dụng các bộ chọn (Selectors) như XPath hoặc CSS để bóc tách thông tin thô: tiêu đề, giá, diện tích, tọa độ, và nội dung mô tả phi cấu trúc.
- Lọc dữ liệu sơ bộ (Validation): Kiểm tra dữ liệu ngay tại nguồn (ví dụ: loại bỏ bài đăng thiếu giá hoặc diện tích) trước khi đẩy vào hệ thống xử lý.

Scrapy cung cấp một hệ sinh thái đầy đủ giúp quản lý việc thu thập dữ liệu lớn một cách liên tục:

- Item Pipeline: Cho phép xử lý dữ liệu ngay sau khi thu thập (làm sạch, kiểm tra trùng lặp) trước khi đẩy sang Kafka.
- Middleware: Hỗ trợ xử lý các vấn đề phức tạp như quản lý Proxy, giả lập User-Agent để tránh bị website chặn.
- Asynchronous Processing: Sử dụng thư viện Twisted để xử lý bất đồng bộ, giúp gửi nhiều yêu cầu HTTP cùng lúc mà không cần chờ đợi, tối ưu hóa tốc độ thu thập.

#### Lợi ích:

- Tốc độ: Nhanh hơn nhiều so với các thư viện như BeautifulSoup hay Selenium do cơ chế bất đồng bộ.
- Quản lý lỗi: Tự động thử lại khi yêu cầu thất bại và quản lý hàng đợi thông minh.
- Mở rộng: Dễ dàng tích hợp thêm các Spider mới khi muốn thu thập thêm từ các nguồn website khác.

#### Nhược điểm:

- Phức tạp hơn so với BeautifulSoup, Selenium đối với người mới bắt đầu.
- Khó xử lý các trang web render bằng JavaScript (SPA) nếu không cài đặt thêm các bộ hỗ trợ như Scrapy-Playwright hoặc Scrapy-Splash.

Nhóm đã cân nhắc giữa các công cụ phổ biến như BeautifulSoup, Selenium và Scrapy, lý do Scrapy được chọn cho bài toán Big Data này là:

- So với BeautifulSoup:
  - Hạn chế của BS4: Chỉ là một thư viện phân tích cú pháp (parser), không có cơ chế quản lý yêu cầu HTTP hay hàng đợi. Đối với dữ liệu lớn, phải tự viết thêm rất nhiều code để quản lý luồng dữ liệu.
  - Ưu thế của Scrapy: Là một Framework hoàn chỉnh. Nó bao gồm cả bộ parser, bộ quản lý Scheduler và bộ ghi log chuyên nghiệp, phù hợp với quy mô dữ liệu lớn của thị trường bất động sản.
- So với Selenium:
  - Hạn chế của Selenium: Selenium giả lập trình duyệt nên tốn rất nhiều tài nguyên CPU/RAM và tốc độ rất chậm. Không phù hợp để cào hàng triệu tin đăng bất động sản mỗi ngày.
  - Ưu thế của Scrapy: Chạy ở cấp độ HTTP Request nên cực kỳ nhẹ và nhanh. Với các trang web có cấu trúc HTML rõ ràng, Scrapy là lựa chọn tối ưu về hiệu suất.
- Khả năng tích hợp Pipeline: Scrapy cho phép viết mã tích hợp trực tiếp với Kafka Producer ngay trong Item Pipeline. Điều này giúp dữ liệu vừa cào về được đẩy ngay lập tức vào luồng xử lý streaming, đảm bảo tính liên tục của hệ thống Big Data.

### 2.3. Airflow

Trong kiến trúc tổng thể của hệ thống Big Data, Apache Airflow đóng vai trò là công cụ điều phối và lập lịch cho toàn bộ pipeline xử lý dữ liệu. Việc sử dụng Airflow giúp hệ thống đảm bảo tính tự động hóa, ổn định và dễ mở rộng, đặc biệt phù hợp với các pipeline dữ liệu có nhiều bước xử lý phụ thuộc lẫn nhau. Nhóm sử dụng Airflow thực hiện các nhiệm vụ:

- Lập lịch tự động: Kích hoạt các Scrapy Spider chạy vào các khung giờ cố định để thu thập dữ liệu mới nhất từ các trang web bất động sản.
- Quản lý sự phụ thuộc: Đảm bảo tính tuần tự của quy trình xử lý dữ liệu.
- Monitoring & Alerting: Theo dõi trạng thái của các pipeline. Nếu một tác vụ bị lỗi, Airflow sẽ gửi cảnh báo và tự động retry theo cấu hình.

Airflow cung cấp một nền tảng quản lý quy trình làm việc dựa trên mã nguồn (Workflow as Code) với các tiện ích cốt lõi:

- DAGs (Directed Acyclic Graphs): Cho phép định nghĩa toàn bộ quy trình xử lý dữ liệu dưới dạng đồ thị có hướng không chu trình bằng ngôn ngữ Python. Điều này giúp quy trình xử lý dữ liệu trở nên minh bạch, dễ bảo trì và có thể quản lý phiên bản.
- Giao diện người dùng (Web UI) trực quan: Cung cấp dashboard để theo dõi trực quan tiến độ của từng pipeline, xem log chi tiết của từng tác vụ, và thực hiện các thao tác thủ công (như chạy lại một pipeline bị lỗi) một cách dễ dàng.

- Operators đa dạng: Cung cấp thư viện các toán tử (Operators) có sẵn để tương tác với các hệ thống khác mà không cần viết nhiều code.

#### Lợi ích:

- Khả năng mở rộng và linh hoạt: Do viết bằng Python, nhóm có thể tùy biến logic phức tạp cho các luồng dữ liệu.
- Backfilling: Tính năng mạnh mẽ cho phép chạy lại quy trình cho các khoảng thời gian trong quá khứ.
- Là tiêu chuẩn công nghiệp nên có tài liệu hỗ trợ phong phú và tích hợp tốt trong nhiều hệ thống.

#### Nhược điểm:

- Độ trễ của Scheduler: Không phù hợp cho các tác vụ yêu cầu thời gian thực với độ trễ thấp.
- Yêu cầu một lượng tài nguyên hệ thống nhất định để duy trì Web Server, Scheduler và Worker hoạt động liên tục.

Airflow giải quyết vấn đề quản lý sự phụ thuộc giữa các tác vụ nhờ cơ chế quản lý trạng thái và dependencies chặt chẽ, giao diện Web UI cho phép nhóm dễ dàng debug và theo dõi sức khỏe hệ thống.

#### 2.4. Kafka

Trong kiến trúc tổng thể, Apache Kafka hoạt động như một lớp đệm trung gian kết nối giữa nguồn dữ liệu và hệ thống lưu trữ đích. Kafka cung cấp khả năng lưu trữ ngắn hạn bền vững. Ngay cả khi consumer hoặc hệ thống đích gặp sự cố, dữ liệu vẫn được bảo toàn trong các Topic và có thể được xử lý lại sau khi hệ thống hồi phục. Nhóm xây dựng Kafka Cluster dưới dạng các pod hoạt động trên hệ thống K8S với kiến trúc phân tán đảm bảo tính chịu lỗi cao:

- Strimzi Operator: Strimzi là một phần mềm chuyên dùng để triển khai, quản lý và vận hành cụm Kafka. Strimzi giúp quản lý hạ tầng như tạo StatefulSet, ConfigMap, Service,... nhờ đó nhóm có thể tập trung hơn vào logic cấp cao như cấu hình cụm Kafka. Việc thiết lập cụm Kafka được thực hiện dễ dàng hơn bằng cách giao tiếp và ra lệnh cho Strimzi thông qua các file yaml.
- Cấu trúc Cluster: 3 Brokers (Broker 0, Broker1, Broker2): Vừa chịu trách nhiệm lưu trữ, quản lý và truyền tải dữ liệu, vừa là controller quản lý siêu dữ liệu. Ở đây, nhóm sử dụng Kraft để quản lý metadata thay vì Zookeeper do Zookeeper không còn được hỗ trợ nữa. Việc sử dụng 3 brokers cho phép phân tải và hỗ trợ cơ chế nhân bản.
- UI for Apache Kafka: Ngoài core kafka, nhóm tích hợp thêm phần mềm UI for Apache Kafka. Đây là một phần mềm của bên thứ 3, phát triển bởi Provectus giúp dễ dàng quản lý apache kafka dưới dạng giao diện app trực quan. Phần mềm này kết nối với cụm Kafka thông qua tên service và cổng mà cụm Kafka cung cấp, từ đó có thể quản lý các vấn đề sức khỏe của cụm, thêm, xóa các topic,...

Quy trình xử lý dữ liệu qua Kafka được thiết kế như sau:

- **Topic Configuration:** Nhóm cài đặt 5 topic để chứa 5 loại dữ liệu thu được: *nhamatpho* - thông tin của nhà phố, *bietthu* - thông tin của biệt thự, *chungcu* - thông tin của chung cư *nharieng* - thông tin của nhà riêng, *dat* - thông tin về loại đất. Topic này được chia thành 2 Partitions, phân phối đều trên các Broker. Việc chia partition giúp tăng tốc độ đọc/ghi song song.
- **Producer:** Producer là chương trình crawl data từ các trang web bất động sản. Dữ liệu crawl được sẽ được xử lý qua và được đẩy vào Kafka thông qua địa chỉ public ip của K8s và cổng expose của cụm kafka.
- **Replication:** Mỗi phiên bản dữ liệu được đẩy vào topic trên 1 broker, tương ứng sẽ có một nhân bản của nó được tạo ra và lưu ở broker khác. Nếu Broker 1 bị sập, Broker 2 sẽ tự động thay thế để phục vụ dữ liệu, đảm bảo không mất mát thông tin thời tiết quan trọng. Việc này được thực hiện hoàn toàn tự động bởi hệ thống quản lý hạ tầng cụm Kafka.
- **Consumer:** 1 Consumer group chịu trách nhiệm lắng nghe 5 Topic, lấy dữ liệu từ các partition, thực hiện ghi xuống MinIO và xử lý và lưu vào Elastic Search.

### Lợi ích

- Decoupling: Tách rời hoàn toàn quá trình thu thập (Producer) và quá trình lưu trữ/phân tích (Consumer), giúp hệ thống linh hoạt
- Reliability : Cơ chế Replication giữa các Broker đảm bảo tính sẵn sàng cao.
- High Throughput: Khả năng xử lý hàng nghìn tin nhắn/giây với độ trễ thấp.

### Nhược điểm

- Tài nguyên hệ thống: Với việc duy trì đồng thời pod Strimzi, 3 pod broker và 1 pod kafka ui, hệ thống máy ảo phải tiêu tốn một lượng không nhỏ RAM và core CPU.

Nhóm quyết định lựa chọn Apache Kafka thay vì các giải pháp khác hoặc việc ghi trực tiếp vào Database vì các lý do sau:

- Đây là công cụ được giới thiệu trong bài giảng. Việc áp dụng và triển khai Kafka có thể giúp các thành viên trong nhóm nắm bắt được các kiến thức trong bài giảng sâu sắc hơn.
- Khả năng mở rộng với Partitioning: Việc cấu hình 5 Topic với 2 partitions là minh chứng rõ nhất cho khả năng xử lý song song. Nếu lượng dữ liệu bất động sản tăng lên nhóm chỉ cần tăng số lượng Consumer và Broker mà không cần thay đổi kiến trúc.
- Hệ sinh thái toàn diện: Kafka là một công cụ được hỗ trợ mạnh mẽ bởi một hệ sinh thái gồm nhiều thành phần và một cộng đồng lớn. Do đó nó đáng tin cậy và luôn có sẵn các thông tin hỗ trợ khi gặp khó khăn.

### 2.5. Spark

Apache Spark đóng vai trò là trung tâm xử lý của toàn bộ hệ thống. Trong dự án này, Spark đảm nhiệm hai vai trò song song:

- Batch Processing: Thực hiện quy trình ETL (Extract - Transform - Load) định kỳ. Spark đọc lượng lớn dữ liệu thô từ MinIO, làm sạch, chuẩn hóa, loại bỏ nhiễu và lưu trữ lại vào Elastic Search để phục vụ phân tích dài hạn.
- Stream Processing: Thông qua module Spark Structured Streaming, hệ thống tiêu thụ dữ liệu trực tiếp từ Kafka topic. Tại đây, Spark thực hiện các phép

### Lợi ích

- Hiệu suất vượt trội: Nhờ cơ chế In-memory Computing, Spark rất nhanh đối với các tác vụ chạy trong RAM
- Unified Engine: Chỉ cần một framework duy nhất để xử lý cả Batch và Streaming, giúp giảm độ phức tạp khi bảo trì mã nguồn.
- Hệ sinh thái phong phú: Tích hợp mượt mà với Kafka (để lấy dữ liệu), MinIO (để lưu trữ), và hỗ trợ mạnh mẽ các ngôn ngữ Python (PySpark), SQL.
- Khả năng chịu lỗi: Cơ chế RDD lineage giúp Spark tự động tính toán lại các phân vùng dữ liệu bị mất nếu một node gặp sự cố.

### Nhược điểm

- Tiêu tốn tài nguyên: Vì xử lý trong RAM, Spark yêu cầu dung lượng bộ nhớ lớn. Chi phí hạ tầng sẽ cao hơn so với các giải pháp xử lý trên đĩa.
- Độ trễ khởi tạo: Thời gian khởi động JVM và phân phối tác vụ khiến Spark không phù hợp cho các truy vấn nhỏ cần phản hồi tức thì (sub-second latency).
- Phức tạp: Với tập dữ liệu nhỏ (vài GB), Spark có thể chậm và cồng kềnh hơn so với dùng Pandas trên một máy đơn.

Giữa các công cụ xử lý dữ liệu như Hadoop MapReduce, Apache Flink hay Pandas, nhóm quyết định chọn Apache Spark vì các lý do sau:

- Phù hợp với mô hình Lambda/Kappa: Dự án yêu cầu cả xử lý dữ liệu lịch sử và xử lý dữ liệu luồng thời gian thực. Spark là công cụ duy nhất cung cấp API thống nhất cho cả hai tác vụ này, giúp nhóm không phải học và duy trì hai hệ thống riêng biệt.
- Khả năng xử lý dữ liệu phi cấu trúc và bán cấu trúc: Dữ liệu đầu vào thường lộn xộn và đa dạng. Spark SQL cung cấp khả năng schema-on-read và các hàm xử lý chuỗi mạnh mẽ, giúp việc chuẩn hóa dữ liệu trở nên dễ dàng hơn nhiều so với viết code MapReduce thuần túy.
- Tốc độ và khả năng mở rộng: Mặc dù hiện tại dữ liệu dự án là mô phỏng, nhưng thiết kế thực tế đòi hỏi khả năng Horizontal Scaling. Spark cho phép dễ dàng thêm node vào cụm cluster để tăng sức mạnh xử lý mà không cần sửa đổi mã nguồn.

### 2.6. MinIO

MinIO được nhóm sử dụng trong việc triển khai Batch Layer trong kiến trúc Lambda. Trong Kubernetes, các Pod có thể sinh ra và mất đi, dữ liệu lưu trong Pod sẽ mất theo. MinIO đóng vai trò là bộ nhớ vĩnh viễn, giúp tách biệt việc "Tính toán" (Spark) và "Lưu trữ" (MinIO), cho phép mở rộng linh hoạt.

#### Lợi ích:

- Tương thích với S3 API: MinIO hiểu được giao thức S3. Do đó khi viết code lưu dữ liệu bằng giao thức S3, MinIO hoàn toàn có thể tương thích và thực hiện được.
- Hiệu năng cao: MinIO được viết bằng ngôn ngữ Go và tối ưu hóa. Nó được coi là Object Storage nhanh nhất thế giới hiện nay, phù hợp cho Spark Big Data.
- Phù hợp với Kubernetes: Cài đặt nhẹ nhàng, quản lý bằng Helm/Operator dễ dàng hơn nhiều so với việc dựng một cụm Hadoop HDFS cồng kềnh.

**Nhược điểm:**

- No Computation : Khác với HDFS đi kèm với MapReduce (tính toán). MinIO chỉ đơn thuần là kho chứa. Khi sử dụng bắt buộc phải cài thêm Spark để xử lý dữ liệu.
- Nhiều file nhỏ: Nếu Spark ghi ra hàng triệu file nhỏ (vài KB), hiệu năng của MinIO sẽ giảm. Do đó cần tối ưu code Spark để ghi ra các file to hơn.

Nhóm chọn MinIO làm Data Lake thay vì các lựa chọn khác như HDFS bởi các lí do:

- Tính đơn giản: MinIO có cơ chế cài đặt rất đơn giản, đặc biệt là trong hệ thống k8s, chỉ cần triển khai 1 pod MinIO. Ngược lại HDFS phải cài đặt nhiều hạ tầng liên quan như NameNode, DataNode,...
- Tính mở rộng: MinIO có thể dễ dàng mở rộng bằng cách thêm các pod mới trong khi HDFS phải thuê thêm hạ tầng phần cứng.
- Chuẩn hóa S3 api: MinIO sử dụng giao thức `s3a://` để gửi dữ liệu, do đó tương thích tốt với các nền tảng cloud như AWS

**2.7. ElasticSearch**

ElasticSearch được nhóm sử dụng để triển khai Serving Layer trong kiến trúc Lambda, là nơi tổng hợp thông tin xử lý luồng (stream processing) và xử lý lô (batch processing) để cung cấp đầu ra cuối cùng vừa đáp ứng nhu cầu độ trễ thấp và nhu cầu chất lượng dữ liệu. ElasticSearch có nhiệm vụ:

- Nhận dữ liệu từ Spark Streaming, đồng thời cũng nhận dữ liệu từ Spark Batching và tổng hợp lại, đưa ra một dataframe tổng hợp.
- Cung cấp dữ liệu cho các truy vấn SQL từ phía người dùng

**Lợi ích**

- Tốc độ cực nhanh: Truy vấn hàng triệu bản ghi chỉ mất vài mili-giây (ms). Nhanh hơn MinIO (vài phút) hay SQL (vài giây) rất nhiều.
- Hỗ trợ Tiếng Việt tốt: Với thông tin các bất động sản bằng tiếng Việt: tiêu đề, mô tả, Elasticsearch có thể hỗ trợ tốt đối với ngôn ngữ này.
- Fuzzy Search: Người dùng gõ sai một vài kí tự, Elasticsearch vẫn hiểu và trả về kết quả đúng.
- Dễ mở rộng: Chỉ cần thêm các pod Elasticsearch vào K8s, chúng sẽ tự động dàn trải dữ liệu.

**Nhược điểm**

- Tốn RAM: ES được viết bằng Java, rất tốn RAM. Đây là thành phần tốn kém nhất trong cụm K8s.
- Không phải Database chuẩn ACID: Không dùng ES để lưu trữ dữ liệu giao dịch quan trọng. Nó có thể bị mất dữ liệu nhỏ trong vài giây.
- Phức tạp khi vận hành: Khi dữ liệu lớn dễ bị lỗi OOM - Out of Memory

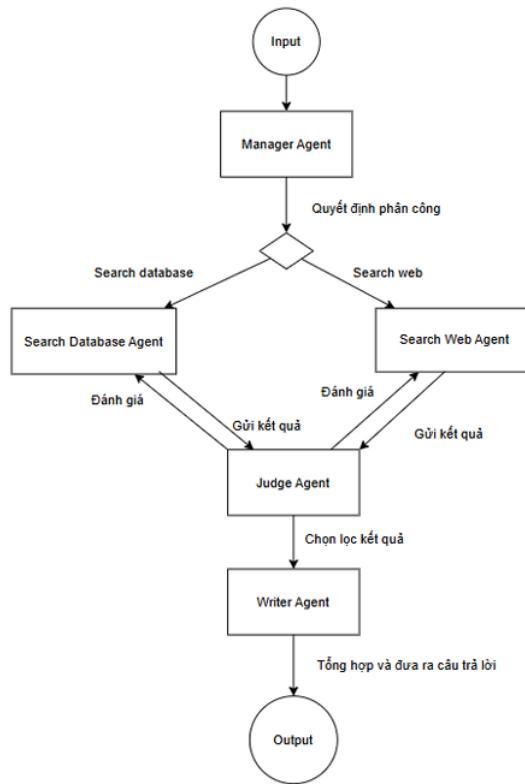
**2.8. Hệ thống đa tác tử (multi-agent)**

Nhằm nâng cao khả năng khai thác dữ liệu bất động sản đã lưu trữ trong Elasticsearch, nhóm triển khai thành phần đa tác tử, mô phỏng tương giữa người dùng và hệ thống dữ liệu, cho phép tiếp nhận yêu cầu bằng ngôn ngữ tự nhiên, phân tích ngữ cảnh, truy xuất dữ liệu từ kho dữ liệu hoặc Internet, và tổng hợp kết quả thành các tư vấn bất động sản phù hợp với từng người dùng cụ thể. Hệ thống được thiết kế dựa trên ba design patterns tiên tiến trong Generative AI:

- Routing (Định tuyến): Phân loại câu hỏi để chuyển đến tác tử chuyên trách.

- Prompt Chaining (Chuỗi nhắc): Kết quả của tác từ này là đầu vào của tác từ khác.
- Evaluator-Optimizer (Đánh giá - Tối ưu): Có cơ chế kiểm tra chất lượng câu trả lời trước khi gửi cho người dùng

Hệ thống bao gồm 5 tác tử chuyên biệt. Hình dưới đây mô tả luồng hoạt động của các tác tử:



**Hình 2. Kiến trúc chi tiết hệ thống đa tác tử**

**Manager Agent:** Tiếp nhận câu hỏi người dùng, phân tích ý định và ra quyết định định tuyến (Routing). Nó sẽ quyết định gọi **Search Database Agent** (nếu hỏi về thông tin nhà cụ thể), **Search Web Agent** (nếu hỏi về pháp lý, tin tức), hoặc cả hai.

**Search Database Agent:** Có vai trò tìm kiếm các bài đăng bất động sản phù hợp với yêu cầu của người dùng. Tác tử này được cung cấp hai hàm để truy cập cơ sở dữ liệu:

- **search\_strict**: Tìm kiếm các bài đăng bất động sản thỏa mãn đồng thời tất cả các tiêu chí mà người dùng đưa ra.
- **search\_flexible**: Tổng hợp các bài đăng thỏa mãn từng tiêu chí riêng lẻ trong yêu cầu của người dùng.

Sử dụng Few-shot Prompting để tự động trích xuất tham số (giá, địa chỉ, số phòng...) và lựa chọn hàm phù hợp. Nếu **search\_strict** rõ ràng, tự động chuyển sang **search\_flexible**.

**Search Web Agent:** chịu trách nhiệm tìm kiếm các thông tin trên Internet liên quan đến các khía cạnh mà cơ sở dữ liệu không bao gồm, như xu hướng thị trường, dự

báo, pháp lý, tiện ích xung quanh,... Tác tử này sẽ trả về một danh sách các thông tin thu thập được từ Internet.

**Judge Agent:** Có vai trò đánh giá chất lượng kết quả tìm kiếm được gửi về từ Search Database Agent và Search Web Agent. Tác tử này được hướng dẫn thông qua prompt dạng Chain-of-Thought kết hợp với Few-shot nhằm loại bỏ các thông tin nhiễu hoặc sai lệch trước khi chuyển tiếp.

**Writer Agent:** Từ các kết quả đã được Judge Agent lựa chọn, tác tử này sẽ tổng hợp, phân tích chuyên sâu và so sánh các thông tin nổi bật, dựa trên hồ sơ và lịch sử của người dùng, tác tử này viết lại câu trả lời cuối cùng dưới dạng văn bản tự nhiên, so sánh các lựa chọn và đưa ra lời khuyên định hướng giá trị.

**Bộ nhớ cho các tác tử:** Để đảm bảo cuộc hội thoại được liền mạch và tự nhiên, các tác tử trên sử dụng một bộ nhớ chung gồm có 2 phần:

- Bộ nhớ ngắn hạn: Lưu trữ cửa sổ trượt gồm 4 tin nhắn gần nhất (2 cặp hỏi-đáp) giữa người dùng và hệ thống giúp hệ thống hiểu các đại từ thay thế.
- Bộ nhớ dài hạn: Sử dụng framework ZepAI để xây dựng đồ thị tri thức 3 cấp độ, giải quyết bài toán quên thông tin trong các hội thoại dài.
- Cơ chế truy xuất: Dựa trên câu hỏi đầu vào của người dùng, thực hiện truy xuất các thông tin: thực thể, fact, cộng đồng và các đoạn tin nhắn gốc dựa trên 3 phương pháp:
  - Cosine Similarity: Tìm kiếm theo ngữ nghĩa (Semantic search).
  - BM25: Tìm kiếm theo từ khóa (Keyword search).
  - BFS (Breadth-First Search): Tìm kiếm theo chiều rộng trên đồ thị (n-hops) để tìm ra các mối liên hệ ẩn.

Các thông tin được tìm kiếm bởi phương pháp 1 và 2 sẽ được tổng hợp và sắp xếp lại (rerank) trước khi mở rộng tìm kiếm bằng phương pháp 3. Ngoài ra, các fact sẽ được sắp xếp theo thời gian, các thông tin này sẽ được dùng làm ngữ cảnh tạo ra câu trả lời.

Nhóm lựa chọn triển khai kiến trúc đa tác tử thay vì mô hình RAG truyền thống vì:

- RAG thông thường chỉ truy xuất các đoạn văn bản tương đồng. Với Bất động sản, người dùng cần sự so sánh, tổng hợp và tư vấn dựa trên nhiều tiêu chí (Giá, Vị trí, Pháp lý, Xu hướng). Chỉ có mô hình Multi-Agent mới có khả năng thực hiện các tác vụ suy luận phức tạp này.
- Khả năng "Nhớ" thông minh với ZepAI: Các Vector Database truyền thống thường làm mất đi mối quan hệ ngữ nghĩa giữa các thông tin. ZepAI với kiến trúc Graph giúp hệ thống hiểu được mối quan hệ nhân quả và sở thích người dùng tốt hơn.
- Tính linh hoạt: Việc tách rời Search Database và Search Web cho phép hệ thống tận dụng tối đa nguồn dữ liệu Big Data do nhóm tự xây dựng, đồng thời không bị lạc hậu thông tin nhờ khả năng tra cứu Web.

## 2.9. Streamlit

Streamlit được lựa chọn làm nền tảng phát triển giao diện người dùng cho dự án nhờ khả năng chuyển đổi linh hoạt các mã lệnh Python thành ứng dụng web tương tác

mà không đòi hỏi chi phí phát triển lớn về Frontend truyền thống. Trong kiến trúc tổng thể của hệ thống, Streamlit đảm nhiệm hai chức năng cốt lõi:

- **Trực quan hóa dữ liệu :** Đây là chức năng chính yếu, biến kho dữ liệu thô không lồ đã thu thập thành những thông tin có giá trị dễ tiếp cận. Hệ thống cung cấp các công cụ thống kê trực quan bao gồm:
  - Biểu đồ cột: Đây là loại biểu đồ dùng để so sánh trực diện các chỉ số quan trọng giữa các quận/huyện trong cùng một tỉnh thành tại một thời điểm nhất định.
  - Biểu đồ đường: Khác với biểu đồ cột (chỉ cho thấy dữ liệu tĩnh), biểu đồ đường tập trung vào sự biến động của thị trường theo dòng thời gian.
  - Bản đồ nhiệt địa lý: Giúp người dùng quan sát toàn cảnh sự phân bố giá cả và mật độ tin đăng theo từng quận/huyện trên phạm vi toàn quốc, hỗ trợ nhận diện nhanh các "điểm nóng" bất động sản.
- **Giao diện tương tác cho hệ thống Đa tác tử:** Vượt ra ngoài một bảng điều hiển tĩnh, Streamlit đóng vai trò là môi trường triển khai và tương tác người-máy cho hệ thống Đa tác tử. Tại đây, ứng dụng tận dụng nguồn dữ liệu sạch đã được xử lý và lưu trữ (trong Elasticsearch) ở các bước trước để phục vụ các tác tử thông minh. Người dùng có thể gửi yêu cầu, và hệ thống sẽ kích hoạt các tác tử để truy xuất dữ liệu, phân tích sâu và phản hồi các thông tin tư vấn cụ thể, tạo nên một luồng xử lý khép kín từ dữ liệu thô đến quyết định đầu tư.

### 3. Luồng dữ liệu và tương tác giữa các thành phần hệ thống

Hệ thống được thiết kế theo kiến trúc xử lý dữ liệu lớn kết hợp giữa xử lý luồng (Stream Layer) và xử lý theo lô (Batch Layer) nhằm đảm bảo vừa cập nhật dữ liệu gần thời gian thực, vừa duy trì tính chính xác và toàn vẹn dữ liệu trong dài hạn. Luồng dữ liệu trong hệ thống được tổ chức theo các giai đoạn liên kết chặt chẽ với nhau như sau:

- **Thu thập dữ liệu:** Apache Airflow đóng vai trò điều phối trung tâm, chịu trách nhiệm lập lịch và quản lý việc thực thi các tác vụ thu thập dữ liệu. Theo lịch định sẵn, Airflow kích hoạt các spider Scrapy để thu thập dữ liệu bất động sản từ nhiều nguồn khác nhau trên Internet. Dữ liệu thu thập được ở giai đoạn này vẫn ở dạng thô và có thể chưa đồng nhất về cấu trúc.
- **Hàng đợi thông điệp Apache Kafka:** đóng vai trò là lớp đệm trung gian giữa quá trình thu thập và xử lý. Kafka giúp tách biệt các thành phần trong hệ thống, đảm bảo khả năng mở rộng, chịu tải cao và tránh mất mát dữ liệu khi lưu lượng tăng đột biến. Đồng thời, Kafka cung cấp dữ liệu cho Stream Layer để phục vụ các yêu cầu xử lý gần thời gian thực.
- **Stream Layer Processing:** Spark Streaming tiêu thụ dữ liệu trực tiếp từ Kafka. Trước khi thực hiện xử lý, dữ liệu thô được lưu trữ vào Data Lake MinIO nhằm đảm bảo khả năng truy vết và tái xử lý khi cần thiết. Sau đó, Spark Streaming tiến hành các bước xử lý nhanh như chuẩn hóa trường dữ liệu, lọc dữ liệu lỗi cơ bản và trích xuất các thuộc tính quan trọng, rồi ghi kết quả vào Elasticsearch để phục vụ truy vấn tức thời.

- **Batch Layer Processing:** Theo chu kỳ định kỳ, Airflow kích hoạt các job Spark Batch, đọc toàn bộ dữ liệu raw đã được lưu trữ trong MinIO. Tại đây, Spark Batch thực hiện các phép xử lý phức tạp hơn như làm sạch dữ liệu sâu, xử lý trùng lặp, phát hiện và sửa các sai lệch có thể phát sinh trong Stream Layer. Kết quả sau xử lý batch sẽ được cập nhật lại vào Elasticsearch, đảm bảo dữ liệu phục vụ truy vấn luôn phản ánh bức tranh đầy đủ và nhất quán.
- **Serving Layer:** Tại Serving Layer, Elasticsearch đóng vai trò là lớp phục vụ truy vấn cho người dùng và các dịch vụ phía trên (ví dụ hệ thống tư vấn đa tác tử). Việc kết hợp dữ liệu từ cả Stream Layer và Batch Layer giúp hệ thống vừa đảm bảo tính cập nhật nhanh, vừa duy trì độ chính xác cao, đáp ứng hiệu quả các yêu cầu phân tích và khai thác dữ liệu bất động sản.
- **Backend Application Layer:** Sử dụng FastAPI làm cầu nối API và vận hành hệ thống Multi-Agent, chịu trách nhiệm truy xuất dữ liệu từ Elasticsearch và xử lý logic nghiệp vụ nền tảng.
- **Frontend Presentation Layer:** Được xây dựng trên Streamlit với trọng tâm là trực quan hóa dữ liệu. Tầng này chuyển đổi dữ liệu thô thành các bản đồ nhiệt và biểu đồ phân tích thị trường chuyên sâu, giúp người dùng dễ dàng quan sát các biến động giá và mật độ tin đăng. Đồng thời, đây cũng là giao diện tích hợp Chatbot để người dùng tương tác trực tiếp với hệ thống đa tác tử.

## CHƯƠNG 4: TRIỂN KHAI HỆ THỐNG

### 1. Môi trường và chiến lược triển khai hệ thống

Hệ thống Big Data của project được thiết kế theo kiến trúc phân tán, bao gồm nhiều thành phần đảm nhiệm các chức năng khác nhau như thu thập dữ liệu, xử lý dữ liệu, lưu trữ, phân tích và trực quan hóa. Các thành phần lưu trữ và xử lý dữ liệu của hệ thống được triển khai theo mô hình container hóa và điều phối bởi Kubernetes (K8S) nhằm đảm bảo tính linh hoạt, khả năng mở rộng và dễ dàng quản lý. Thành phần thu thập dữ liệu được triển khai local để đảm bảo tài nguyên hạ tầng. Hệ thống được triển khai trên một máy tính cá nhân và 4 instance với dịch vụ EC2 trên nền tảng Amazon Web Services (AWS), phù hợp với quy mô của bài tập lớn và tài nguyên cho phép.

#### 1.1. Môi trường triển khai phần cứng và hệ điều hành

Máy tính cá nhân: Hệ điều hành Windows, nhưng sử dụng wsl để chạy giả lập môi trường Linux:

- Hệ điều hành: Ubuntu 20.04 LTS
- Kiến trúc: x86\_64
- RAM: 8GB
- Vi xử lý: Intel Core i7-1165G7, 4 nhân xử lý

Máy tính cá nhân được sử dụng để:

- Triển khai chương trình crawler lấy dữ liệu từ trang web
- Chạy airflow để lập lịch điều khiển hệ thống
- Triển khai Frontend (Streamlit) và Backend phục vụ tương tác người dùng.
- Chạy các Agent xử lý yêu cầu người dùng và điều phối truy vấn dữ liệu.
- Phát triển, kiểm thử và giám sát hệ thống trong giai đoạn xây dựng project.

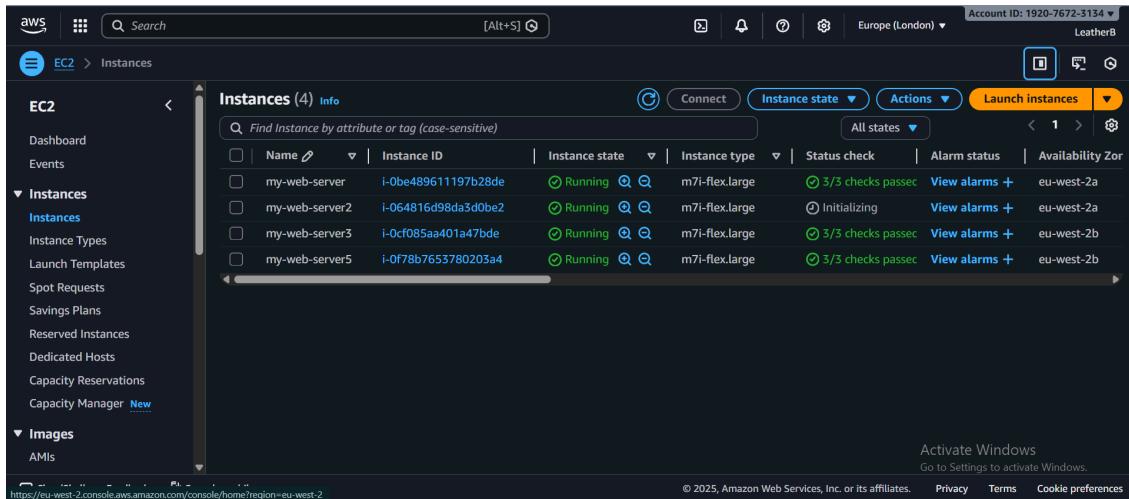
4 instance trên AWS là loại *m7i-flex* thuộc dòng *m* cung cấp hiệu suất ổn định. Cấu hình cụ thể:

- Hệ điều hành: Ubuntu 20.04 LTS
- Kiến trúc: x86\_64
- RAM: 8GB
- CPU: 2 vCPUs ( 4GiB RAM/ 1CPU)

Các máy ảo này được sử dụng để triển khai:

- Apache Kafka: thu thập và truyền tải dữ liệu dòng giữa các thành phần.
- MinIO: hệ thống lưu trữ đối tượng, đóng vai trò lưu trữ dữ liệu thô và dữ liệu đã xử lý.
- Elasticsearch: lưu trữ và lập chỉ mục dữ liệu phục vụ tìm kiếm nhanh cho backend và chatbot.
- Apache Spark: lấy dữ liệu từ nguồn( Kafka, MinIO) và đẩy đến đích( MinIO, Elasticsearch)

Do điều kiện tài nguyên và hạn mức hạn chế, nhóm chỉ có thể tạo tối đa 8 core CPU, vì vậy 4 máy ảo trên AWS EC2.



**Hình 3.** Các instance được triển khai trên EC2

## 1.2. Chuẩn bị môi trường triển khai

Trước khi triển khai hệ thống, cần thiết lập các công cụ và tài nguyên sau:

- Docker: nhóm cần có Docker để đóng gói các ứng dụng spark thành các image, phục vụ cho việc triển khai thành các pod trên Kubernetes.
- GitHub: quản lý mã nguồn và version control.
- AWS EC2: một hệ sinh thái của AWS cung cấp các dịch vụ: các máy ảo( EC2 Instance), dịch vụ lưu trữ ( Elastic Block Store - EBS), dịch vụ IP công cộng (Elastic IP), ... Cần phải có tài khoản AWS với đủ hạn mức để sử dụng hệ sinh thái EC2.

Việc chuẩn bị đầy đủ môi trường giúp đảm bảo quá trình triển khai diễn ra ổn định và nhất quán.

## 1.3. Chiến lược triển khai hạ tầng

Hệ thống được triển khai theo chiến lược:

- Phân tách rõ ràng tầng thu thập – xử lý – lưu trữ – hiển thị.
- Mỗi thành phần hoạt động độc lập nhưng được kết nối thông qua Kafka và các Agent.
- Kết hợp triển khai trên cả hạ tầng AWS EC2 và máy tính cá nhân để đảm bảo tài nguyên hệ thống. Đối với các dịch vụ trên AWS EC2, triển khai từng dịch vụ dưới dạng container để dễ dàng kiểm soát và mở rộng.

Việc sử dụng Kubernetes cho phép triển khai hệ thống theo hướng cloud-native, phù hợp với các yêu cầu của hệ thống Big Data hiện đại.

## 2. Triển khai hạ tầng Kubernetes

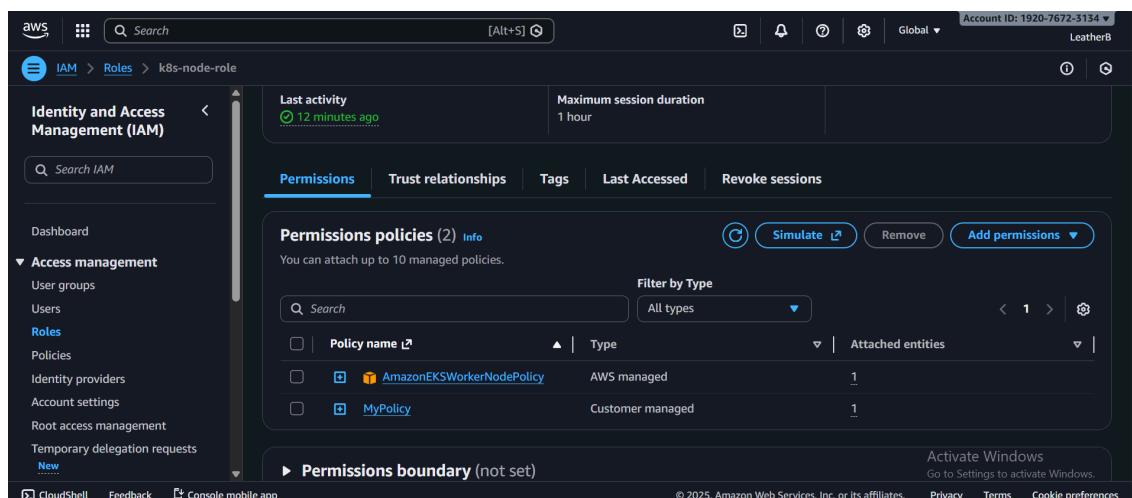
Kubernetes là hạ tầng của toàn bộ hệ thống, là nơi đảm bảo các dịch vụ luôn được hoạt động ổn định bằng cách tái lập lịch nếu dịch vụ bị lỗi, chết; phân bổ, quản lý tài nguyên cho cân bằng và hợp lý, đảm bảo các dịch vụ có đủ tài nguyên để hoạt động ổn định; đảm bảo giao tiếp mạng giữa các pod.

Hạ tầng Kubernetes hoàn toàn được triển khai trên các máy ảo EC2 của AWS.

- Trên các máy ảo cần phải cài một container runtime để có thể chạy được các dịch vụ đã đóng gói thành container. Ở đây nhóm chọn sử dụng *containerd* làm

container runtime thay vì docker vì *containerd* nhẹ hơn, giúp tiết kiệm tài nguyên hơn và cũng được tối ưu cho K8s.

- Cài đặt Kubernetes cores:
  - Cài đặt *kube-scheduler* để lập lịch và điều phối pod
  - Cài đặt *kubeadm*, đây là công cụ để khởi tạo và xây dựng hạ tầng của cụm K8s
  - Cài đặt *kubectl* để giao tiếp với cụm K8s, thông qua công cụ *kubectl*, người dùng có thể thực hiện các thao tác như xem node, xem các tài nguyên, quản lý các tài nguyên, quản lý trạng thái,...
- Để đảm bảo các pod có thể giao tiếp mượt mà với nhau, khi một pod nhận được 1 gói tin với ip không phải của mình, thay vì hủy gói tin, nó nên chuyển tiếp sang pod khác. Do đó, cần bật *ip\_forward* để biến một node thành một router
- Bật *br\_netfilter* để đảm bảo tất cả các gói tin đều phải hoạt động ở tầng IP, giúp Kubernetes quản lý gói tin dễ dàng hơn.
- Để các pod có thể thực sự giao tiếp với nhau, cần cài cho cụm K8s 1 mạng phủ. Trong dự án này, nhóm lựa chọn cài đặt *flannel* làm mạng phủ. *Flannel* cung cấp cho các pod trong 1 node 1 IP ảo trong cùng 1 mạng LAN, do đó các pod có thể gửi trực tiếp gói tin qua nhau. Thao tác cài đặt mạng phủ chỉ thực hiện trên master node, các worker node sẽ tự động được cài đặt khi tham gia cụm.
- Các pod trên Kubernetes, đặc biệt là các pod thực hiện nhiệm vụ lưu trữ dữ liệu đều cần sử dụng dịch vụ lưu trữ EBS của hệ sinh thái EC2. Các pod sẽ gọi API đến AWS và yêu cầu tạo phân vùng lưu trữ mới. Do đó cần phải tạo vai trò IAM có quyền tạo phân vùng lưu trữ từ dịch vụ EBS và gán nó cho tất cả các instance.



*Hình 4. Tạo vai trò IAM với quyền truy cập dịch vụ EBS*

Việc cài đặt các thành phần cần thiết là hết sức quan trọng trong việc triển khai hạ tầng Kubernetes, nó đảm bảo xây dựng được 1 cụm k8s hoạt động ổn định, khỏe mạnh. Sau khi các thành phần đã được cài đặt, thao tác tạo cụm sẽ được thực hiện trên master node. Và các worker node sẽ tham gia vào cụm bằng 1 lệnh cung cấp bởi master node. Với 4 máy ảo, nhóm đang triển khai 1 master node và 3 worker node.

### 3. Triển khai thành phần thu thập và tiền xử lý dữ liệu

Thành phần thu thập và tiền xử lý dữ liệu đóng vai trò là điểm khởi đầu của toàn bộ pipeline Big Data, chịu trách nhiệm thu thập dữ liệu thô từ các nguồn trực tuyến, chuẩn hóa thông tin và loại bỏ dữ liệu nhiễu trước khi đưa vào hệ thống xử lý phía sau. Trong hệ thống này, thành phần thu thập và tiền xử lý được triển khai gồm hai dịch vụ chính:

- Dịch vụ thu thập dữ liệu: Scrapy Spider thực hiện trích xuất dữ liệu từ các trang web bất động sản.
- Dịch vụ lập lịch: sử dụng Apache Airflow để tự động hóa và điều phối quá trình thu thập dữ liệu theo lịch trình định sẵn.

#### 3.1. Dịch vụ thu thập dữ liệu

Dữ liệu được thu thập từ các bài đăng bán bất động sản thuộc bốn loại chính: nhà phố, nhà riêng, chung cư và biệt thự, trên hai website bất động sản là [bds.com.vn](http://bds.com.vn) và [bds68.com.vn](http://bds68.com.vn)

Mỗi website được xử lý bởi một Spider riêng biệt, được xây dựng dựa trên framework Scrapy, cho phép tự động truy cập, thu thập và trích xuất dữ liệu từ các bài đăng một cách hiệu quả và có khả năng mở rộng.

##### a) Cấu hình và tham số của Spider

Các Spider được thiết kế linh hoạt thông qua các tham số cấu hình, bao gồm:

- min\_page: trang phân trang bắt đầu crawl (mỗi trang chứa khoảng 20 liên kết bài đăng).
- max\_page: trang phân trang kết thúc crawl.
- estate\_type: loại bất động sản cần thu thập (nhà phố, nhà riêng, chung cư, biệt thự).
- kafka\_bootstrap\_servers: địa chỉ cụm Kafka dùng để gửi dữ liệu sau khi xử lý.
- kafka\_topics: các topic Kafka mà Spider sẽ gửi dữ liệu tới.

Việc cấu hình tham số giúp hệ thống dễ dàng mở rộng phạm vi crawl mà không cần thay đổi mã nguồn.

##### b) Trích xuất dữ liệu và cấu trúc Items

Sau khi truy cập từng bài đăng, Spider thu thập nội dung HTML và phân tích các thẻ chứa thông tin cần thiết. Các trường dữ liệu được định nghĩa trong lớp Items bao gồm:

- title: tiêu đề bài đăng.
- description: nội dung mô tả chi tiết.
- price: giá bất động sản.
- square: diện tích.
- estate\_type: loại bất động sản.
- address: địa chỉ bất động sản.
- post\_date: ngày đăng bài.
- contact\_info: thông tin liên hệ, bao gồm:
  - phone: số điện thoại.
  - name: tên người liên hệ.
- extra\_infos: các thông tin bổ sung:
  - no\_floors: số tầng.

- *no\_bedrooms*: số phòng ngủ.
- *no\_bathrooms*: số phòng tắm.
- *front\_road*: lộ giới.
- *front\_face*: mặt tiền.
- *yo\_construction*: năm xây dựng.
- link: liên kết gốc của bài đăng.

Việc chuẩn hóa cấu trúc dữ liệu ngay từ bước thu thập giúp định nghĩa schema rõ ràng hơn, đơn giản hơn, giảm đáng kể chi phí xử lý ở các giai đoạn sau.

### 3.2. Tiền xử lý dữ liệu

Dữ liệu sau khi trích xuất được đưa qua pipeline tiền xử lý, bao gồm hai bước chính: hiệu chỉnh địa chỉ và loại bỏ bài đăng trùng lặp, nhằm nâng cao chất lượng dữ liệu đầu vào cho hệ thống Big Data.

#### a) Hiệu chỉnh địa chỉ

Để đảm bảo thông tin địa chỉ được thống nhất, đầy đủ và chính xác theo các cấp hành chính (tỉnh/thành phố, quận/huyện, phường/xã), hệ thống thực hiện hiệu chỉnh địa chỉ theo các bước sau:

- Loại bỏ tiền tố hành chính: Xóa các tiền tố như Tỉnh, Thành phố, Quận, Phường nhằm chuẩn hóa chuỗi địa chỉ đầu vào.
- Xác định cấp hành chính:
  - Sử dụng tệp JSON định nghĩa sẵn các đơn vị hành chính để đổi chiểu và trích xuất thông tin quận/huyện, phường/xã.
  - Trong trường hợp không xác định được chính xác, pipeline sử dụng thư viện geoapivietnam kết hợp kỹ thuật fuzzy matching để suy đoán đơn vị hành chính phù hợp dựa trên tỉnh/thành phố và chuỗi địa chỉ đầy đủ.
- Khôi phục tiền tố chuẩn hóa: Tái cấu trúc địa chỉ theo định dạng chuẩn, ví dụ: Phường Thanh Xuân Trung, Quận Thanh Xuân, Hà Nội.
- Cập nhật trường địa chỉ: Ghi đè trường address trong item bằng địa chỉ đã được chuẩn hóa.

#### b) Loại bỏ bài đăng trùng lặp

Để hạn chế dữ liệu trùng lặp, hệ thống sử dụng pipeline loại bỏ bài đăng trùng lặp dựa trên xử lý ngôn ngữ tự nhiên (NLP) với mô hình TF-IDF, bao gồm các bước:

- Tải checkpoint mô hình TF-IDF đã được huấn luyện trước. Chuyển tiêu đề và mô tả của bài đăng thành vector TF-IDF. So sánh bài đăng mới với danh sách các bài đăng đã xử lý (tối đa 4000 bài) bằng độ đo cosine similarity.
- Xác định trùng lặp dựa trên hai tiêu chí:
  - Độ tương đồng tiêu đề > 0.95
  - Độ tương đồng mô tả > 0.99
- Nếu thỏa mãn một trong hai tiêu chí, bài đăng sẽ bị loại bỏ. Ngược lại, bài đăng được thêm vào danh sách bài đăng đã xử lý. Để tối ưu bộ nhớ, danh sách này được giới hạn 4000 bài và áp dụng chiến lược FIFO, giữ lại 75% các bài đăng gần nhất.

### 3.3. API thu thập dữ liệu

Hệ thống cung cấp API thu thập dữ liệu được xây dựng bằng FastAPI, cho phép kích hoạt quá trình crawl một cách linh hoạt thông qua phương thức POST với các tham số:

- min\_page: Trang bắt đầu
- max\_page: Trang kết thúc
- estate\_type: Loại bất động sản

API đóng vai trò là cầu nối giữa dịch vụ lập lịch và dịch vụ thu thập dữ liệu.

### 3.4. Dịch vụ lập lịch thu thập dữ liệu

Hệ thống sử dụng Apache Airflow để tự động gọi API thu thập dữ liệu theo lịch trình hàng giờ, với chiến lược:

- Chung cư và nhà riêng: thu thập 100 bài đăng mới nhất mỗi giờ.
- Biệt thự và nhà phố: thu thập 60 bài đăng mới nhất mỗi giờ.

Dịch vụ thu thập dữ liệu và Airflow được triển khai local trên cùng 1 máy tính cá nhân, đảm bảo khả năng giao tiếp ổn định và dễ dàng triển khai

```

run_local_crawler
Essais de tâche
1 ✓ 2 ✓ 4 ✘ 5 ✘ 6 ✘ 8 ✘ 9 ⏱
Tous les niveaux de... Paramètres des journaux ⏱

110 [2025-12-21 19:33:33] INFO -     'full_address': 'Đường Cử Xã Đồng Tiến , Phường 14 , Quận 10 , Hồ Chí Minh', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
111 [2025-12-21 19:33:33] INFO -     'ward': None, source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
112 [2025-12-21 19:33:33] INFO -     'ward': 'None', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
113 [2025-12-21 19:33:33] INFO -     'ward': None, source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
114 [2025-12-21 19:33:33] INFO -     'contact_info': {'name': 'Minh Quân', 'phone': ['0938753xxx']}, source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
115 [2025-12-21 19:33:33] INFO -     'description': 'GÒNG LÀI BÁN GẤP CĂN NHÀ NGAY KHU CỦ XÃ ĐỒNG TIẾN , P. 14 , QUẬN 10', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
116 [2025-12-21 19:33:33] INFO -     'ward': 'QUẬN 10', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
117 [2025-12-21 19:33:33] INFO -     'estate_type': 'Nhà phố', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
118 [2025-12-21 19:33:33] INFO -     'extra_infos': {'Pháp lý': 'Sổ đỏ/ Sổ hồng', 'Số Phòng Ngủ': '4', 'Số Phòng Tắm': '6', 'Số Tầng': '3'}, source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
119 [2025-12-21 19:33:33] INFO -     'link': 'https://bds68.com.vn/ban-nha-mat-tien/ho-chi-minh/quan-10/duong-cu-xa-dong-tien-can-ban-nha-ngay-khu-cu-xa-dong-tien-quan-10-gia-4-ty-560-trieu-5x19...', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
120 [2025-12-21 19:33:33] INFO -     'post_date': '2025/12/21', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
121 [2025-12-21 19:33:33] INFO -     'post_id': '29576086', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
122 [2025-12-21 19:33:33] INFO -     'price': 4560000000.0, source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
123 [2025-12-21 19:33:33] INFO -     'square': 95.0, source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
124 [2025-12-21 19:33:33] INFO -     'title': 'Căn bán nhà ngay khu cư xá đồng tiến , quận 10 giá 4 tỷ 560 triệu ( ' source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
125 [2025-12-21 19:33:33] INFO -     'unit': '5x19') ngay thpt nguyên khuyến', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
126 [2025-12-21 19:33:33] INFO -     'unit': '5x19', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
127 [2025-12-21 19:33:33] INFO -     'unit': '5x19', source=airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook loc=subprocess.py:99
128 [2025-12-21 19:33:33] INFO -

```

Hình 5. Airflow lập lịch chạy crawler

## 4. Triển khai thành phần xử lý và lưu trữ dữ liệu

Thành phần xử lý và lưu trữ dữ liệu đóng vai trò trung tâm trong kiến trúc hệ thống, chịu trách nhiệm xử lý dữ liệu thời gian thực (stream processing) kết hợp với xử lý theo lô (batch processing), đồng thời tổ chức lưu trữ và hỗ trợ truy vấn dữ liệu phục vụ các tác vụ phân tích và tìm kiếm.

Hệ thống được triển khai trên 4 máy ảo AWS EC2, sử dụng Kubernetes (K8S) để điều phối tài nguyên và triển khai các dịch vụ một cách linh hoạt, đảm bảo khả năng mở rộng và tính sẵn sàng cao.

Hệ thống xử lý và lưu trữ dữ liệu gồm hai phần chính:

- Xử lý dữ liệu: sử dụng PySpark
- Lưu trữ và truy vấn dữ liệu:
  - Sử dụng Kafka làm hàng đợi thông điệp tạm thời
  - MinIO được sử dụng làm kho lưu trữ dữ liệu lâu dài
  - Sử dụng Elasticsearch làm hệ quản trị lưu trữ dữ liệu dạng tài liệu và truy vấn, phân tích dữ liệu bằng thư viện pandas của Python.

#### 4.1. Lưu trữ dữ liệu

Các thành phần lưu trữ dữ liệu của hệ thống khi hoạt động sẽ cần sử dụng dịch vụ EBS của AWS EC2 để tạo một kho chứa dữ liệu. Do đó trước khi cài đặt các thành phần, cần đảm bảo rằng instance phải được gắn với 1 vai trò IAM có quyền truy cập dịch vụ lưu trữ. Những thành phần chịu trách nhiệm lưu trữ dữ liệu bao gồm:

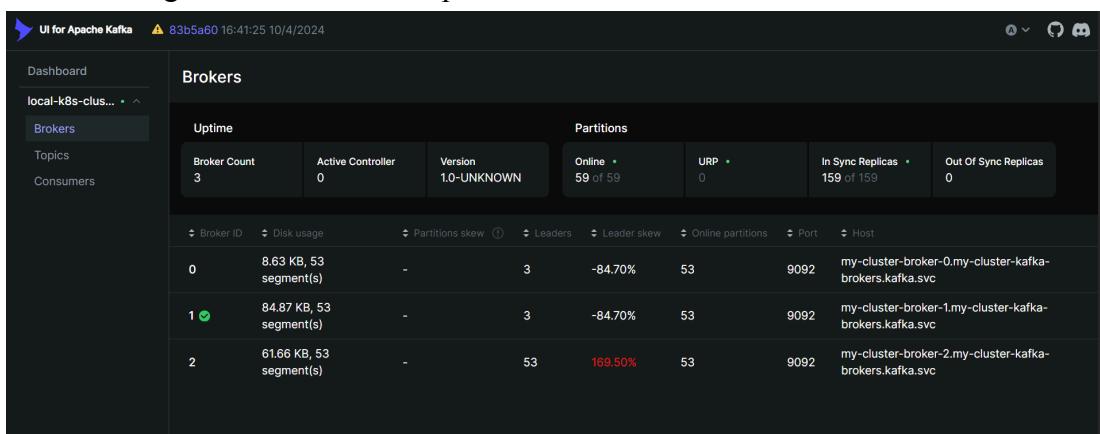
- Kafka - Lưu trữ dữ liệu tạm thời
- MinIO - Lưu trữ dữ liệu lâu dài
- Elasticsearch - Lưu trữ và cung cấp dữ liệu cuối cùng

##### a) Kafka:

Kafka là queue lưu trữ tạm thời để đảm bảo một đầu ra tin cậy cho producer và một đầu vào tin cậy cho consumer.

Cấu hình cụm kafka:

- Số lượng broker: 3 brokers
- Số lượng topic: 5 topic: *nhamatpho, bietthu, chungcu, nharieng, dat*
- Số lượng nhân bản cho các topic: 2



Hình 6. Các broker trong cụm Kafka

##### Triển khai cụm:

- Strimzi operator được cài đặt bằng helm và triển khai dưới dạng 1 pod chạy liên tục để giám sát sức khỏe của cụm kafka và giám sát yêu cầu từ người dùng. Khi strimzi được cài đặt, nó sẽ định nghĩa cho một loạt các tài nguyên đặc thù của Kafka như *KafkaNodepool, KafkaTopic,...*
- Các broker sẽ được khởi tạo và chạy dưới dạng các pod, khi các broker bị chết, strimzi pod sẽ tự phát hiện và tái lập lịch chạy lại pod mới.
- Cụm kafka sau khi chạy ổn định sẽ cung cấp một service để giao tiếp với các pod khác. kafka-ui sẽ được triển khai thành 1 pod, lấy dữ liệu từ service kafka để phục vụ trực quan hóa.

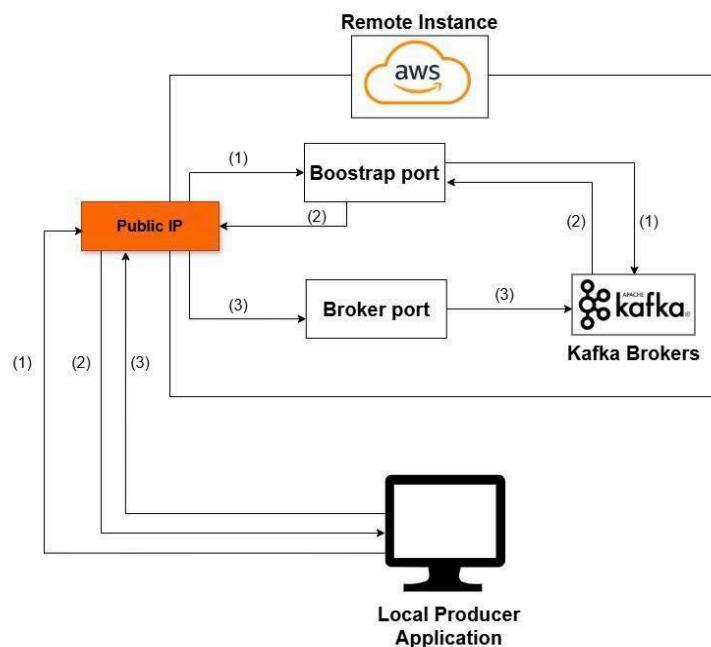
##### Triển khai luồng dữ liệu:

- Producer: producer của kafka là chương trình crawler chạy ở máy tính cá nhân. Sau khi producer crawl, dữ liệu sẽ được đẩy vào Kafka.
- Consumer: consumer là chương trình pyspark, tiêu thụ dữ liệu từ Kafka.
- Trên thực tế, việc gửi dữ liệu từ crawler trên máy tính cá nhân đến kafka trên các instance EC2 không đơn giản. Nhóm đã thực hiện cấu hình như sau để cho phép giao tiếp thành công:

- Expose công ClusterIP ứng với bootstrap service của Kafka (được sử dụng để giao tiếp nội bộ trong cụm Kubernetes) thành 1 cổng NodePort public cho phép dữ liệu đến từ ngoài cụm Kubernetes.
- Sử dụng dịch vụ Elastic IP để gán 1 IP cố định cho instance trong cụm, nhờ đó crawler có thể biết chính xác nên gửi đến đâu.
- Thêm luật cho tường lửa của các máy ảo: cho phép tất cả các truy cập TCP từ bên ngoài.
- Sau khi cấu hình, crawler có thể gửi dữ liệu đến ip public và cổng expose

Quá trình gửi dữ liệu đến các topic được trình bày trong hình dưới đây:

- (1) - Crawler sẽ gửi một yêu cầu giao tiếp đến public IP với cổng expose ứng với bootstrap service của Kafka
- (2) - Các broker thông qua cổng bootstrap sẽ nhận yêu cầu và gửi một thông điệp ngược lại cho crawler. Thông điệp chứa thông tin về cổng riêng để giao tiếp với broker, thay vì sử dụng cổng bootstrap.
- (3) - Về sau, crawler sẽ gửi dữ liệu đến các topic trên các broker kafka thông qua cổng này

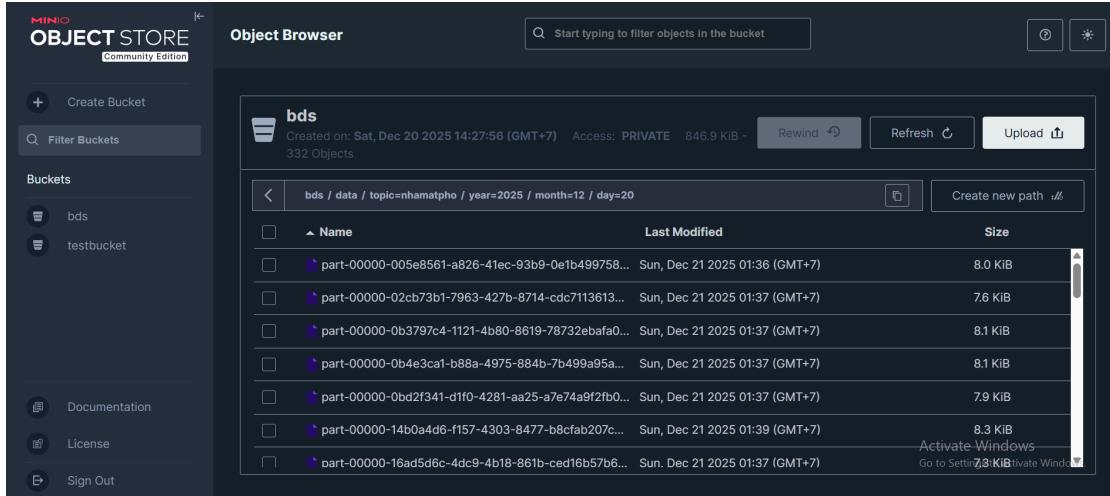


**Hình 7. Luồng giao tiếp của crawler local và remote kafka**

### b) MinIO

MinIO là nơi lưu trữ dữ liệu lâu dài bền vững cho hệ thống. MinIO cung cấp cơ chế cài đặt đơn giản hơn Kafka: không cần phải cài đặt operator, có thể trực tiếp tạo pod từ image trên docker hub của minio. Tuy nhiên, nhóm triển khai tài nguyên Deployment của Kubernetes, và Deployment sẽ chạy pod thay vì trực tiếp triển khai 1 pod MinIO. Điều này mang lại lợi ích trong quản lý: khi pod chết, deployment có thể nấm bắt được và tạo lại một pod mới; trong khi nếu chỉ triển khai 1 pod, khi pod bị lỗi và bị sập, nó không thể tự tái khởi động lại.

MinIO không cung cấp một service để giao tiếp với các pod khác như bootstrap service của Kafka, do đó nhóm đã tạo một service cho MinIO bằng cách triển khai tài nguyên Service của Kubernetes và cho nó trỏ đến pod MinIO. Service được tạo có một cổng để giao tiếp nội bộ trong cụm Kubernetes và một cổng NodePort để có thể truy cập được giao diện MinIO từ bên ngoài cụm.



Hình 8. Giao diện MinIO quản lý dữ liệu trong bucket

MinIO sau khi triển khai có cung cấp một giao diện quản lý, nhờ vào service đã gắn, nhóm có thể quản lý MinIO thông qua giao diện web. Giao diện web cho phép nhóm tạo và quản lý các *bucket*, trong dự án này, nhóm tạo 1 *bucket* *bds* để chứa toàn bộ dữ liệu thô.

### Triển khai luồng dữ liệu

- Luồng dữ liệu đưa vào MinIO xuất phát từ Kafka, được vận chuyển bởi code PySpark. Dữ liệu thô được lưu trong MinIO bắt nguồn từ Kafka thay vì trực tiếp từ crawler là bởi các lí do:
  - Sự phức tạp của giao tiếp từ xa: Để crawler chuyển dữ liệu từ máy tính cá nhân lên Kafka trên EC2 instance, nhóm đã phải thực hiện nhiều cấu hình phức tạp. Trong khi việc nhận dữ liệu từ Kafka nội bộ trong cụm Kubernetes lại đơn giản hơn nhiều
  - Tính sẵn sàng của dữ liệu: Crawler chuyển dữ liệu trực tiếp vào MinIO có thể gặp rào cản về tốc độ tiêu thụ dữ liệu. Crawler(producer) có thể có tốc độ sản sinh dữ liệu lớn hơn tốc độ ghi của consumer(MinIO), khi đó, hiệu suất lưu trữ của hệ thống lại phụ thuộc vào thành phần chậm nhất, gây tắc nghẽn hệ thống. Kafka sinh ra để giải quyết việc này: với nhiều broker phân tán, Kafka đóng vai trò là kho chứa dữ liệu tạm thời, các dữ liệu từ crawler sẽ được lưu tạm vào đây, đảm bảo tốc độ của hệ thống. Và bắt cứ khi nào MinIO ghi xong thì luôn có dữ liệu sẵn sàng để ghi tiếp.
- Dữ liệu trong MinIO là dữ liệu thô của toàn bộ lịch sử, được tiêu thụ bởi chương trình PySpark chịu trách nhiệm xử lý lô dữ liệu.

### c) Elasticsearch

```

{
  "took": 1,
  "shards": 1,
  "hits": [
    {
      "index": "nhapho_index",
      "id": "29572308",
      "score": 1,
      "source": {
        "estate_type": "Nhà phố",
        "address": {
          "district": "Cầu Giấy",
          "full_address": "Đường Hoàng Quốc Việt , Phường Nghĩa Đô , Cầu Giấy , Hà Nội",
          "province": "Hà Nội",
          "ward": "Nghĩa Đô"
        },
        "contact_info": {
          "name": "Nguyễn Thu Hiền",
          "phone": [
            "0962111xxxx"
          ]
        },
        "description": "CĂN BẢN NHÀ C4 PHỐ HOÀNG QUỐC VIỆT , NGHĨA ĐÔ , CẦU GIẤY .",
        "extra_infos": [
          "no_bedrooms": 1,
          "no_bathrooms": 1,
          "front_road": 12,
          "front_face": 10,
          "no_floors": 1
        ],
        "post_date": "2025/12/21",
        "post_id": "29572687",
        "price": 6500000000,
        "square": 251,
        "title": "Bán gấp căn giá rẻ mặt tiền Sư Thiện Chiểu , Phường Võ Thị Sáu , Quận 3 .",
        "kafka_timestamp": 1766255766337,
        "topic": "nhapho",
        "year": 2025,
        "month": 12,
        "day": 20,
        "id": "29572687",
        "price/square": 258964143.426295,
        "created_at": "2025/12/23 17:19:16"
      }
    }
  ]
}

```

**Hình 9.** Giao diện hệ thống Elasticsearch

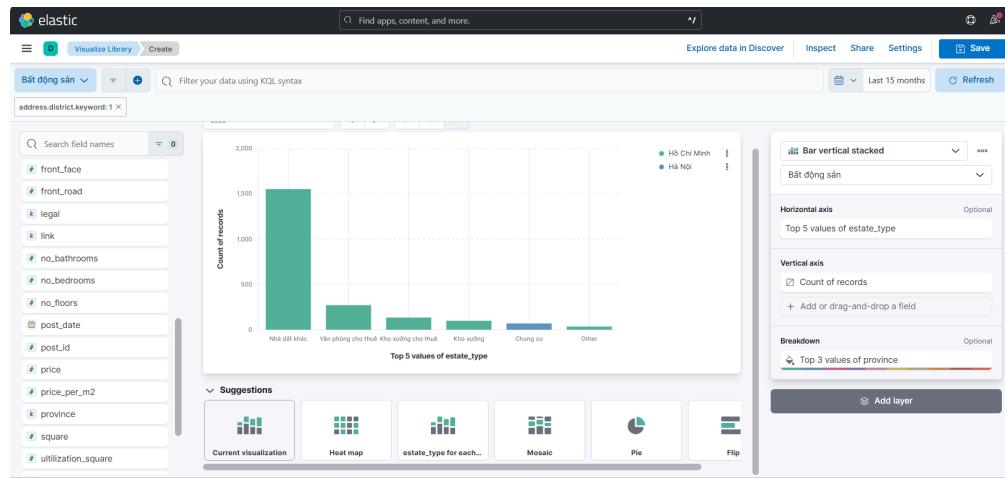
Elasticsearch là nơi lưu trữ dữ liệu cung cấp cho người dùng cuối.

### Triển khai Elasticsearch:

- Cài đặt Elasticsearch CRD (Custom Resource Definition): CRD giúp định nghĩa một loại tài nguyên mới cho K8s, ở đây là tài nguyên Elasticsearch. Đối với Kafka, CRD được cài một cách tự động khi pod strimzi được triển khai.
- Operator: Elasticsearch operator được triển khai thành một pod chạy liên tục để quản lý, giám sát hoạt động của pod Elasticsearch.
- Elasticsearch được triển khai thành một pod trong hệ thống k8s. Do Elasticsearch được tối ưu cho việc truy vấn dữ liệu bằng cách tận dụng sức mạnh của RAM, nên việc chạy pod
- Elasticsearch rất tốn tài nguyên hệ thống. Ở đây, để tiết kiệm tài nguyên, nhóm chỉ triển khai 1 pod Elasticsearch với giới hạn:
  - RAM tối đa được sử dụng là: 2 GiB
  - Lưu trữ được cấp phát: 10GiB

### Triển khai luồng dữ liệu

- Elasticsearch nhận dữ liệu từ chương trình PySpark xử lý luồng và xử lý lô. Các chương trình này sẽ tạo các index trong Elasticsearch và ghi các bản ghi vào đúng index tương ứng với các loại bất động sản.



**Hình 10.** Giao diện Kibana giúp người dùng theo dõi Elasticsearch hiệu quả hơn

#### 4.2. Xử lý dữ liệu

Các pod Spark liên tục lắng nghe và đọc dữ liệu từ các topic Kafka: *nhamatpho, bietthu, chungcu, nharieng, dat*

Dữ liệu đầu vào từ Kafka là các bản ghi JSON, được chuyển đổi thành DataFrame có cấu trúc.

##### a) Xử lý dữ liệu theo luồng (Stream Processing)

Spark Streaming được sử dụng để xử lý dữ liệu ngay khi dữ liệu được sinh ra từ hệ thống thu thập.

##### Triển khai xử lý dữ liệu theo luồng:

Nhóm triển khai code PySpark thực hiện nhiệm vụ:

- Đọc dữ liệu từ Kafka topic
- Thêm id cho các bản ghi dữ liệu
- Ghi dữ liệu vào data lake.
- Thực hiện các bước xử lý chính
- Ghi dữ liệu xử lý vào Elasticsearch

Sau đó code được đóng gói thành image và đẩy lên Docker Hub. Image sau đó sẽ được kéo về và chạy như một container bằng container runtime trên cụm Kubernetes. Nhóm triển khai code trên hệ thống K8s bằng tài nguyên Deployment của K8s thay vì sử dụng trực tiếp tài nguyên Pod. Lý do là bởi tài nguyên Deployment giúp theo dõi giám sát pod, quản lý vòng đời của pod, lập lịch cho pod tái khởi động nếu pod chết.

Nhóm triển khai nhiệm vụ đẩy dữ liệu thô từ Kafka vào MinIO ngay trong 1 pod thay vì triển khai 2 pod với 2 nhiệm vụ riêng: 1 pod xử lý streaming và 1 pod đẩy dữ liệu vào datalake (MinIO) là bởi:

- Tận dụng code đã viết: Nếu triển khai 2 pod, phải viết 2 file code nhưng lại đều thực hiện logic đọc dữ liệu từ Kafka, định nghĩa schema và thêm id.
- Tiết kiệm tài nguyên: Khi triển khai 2 pod thì phải đóng gói 2 lần. Khi đó, các môi trường sẽ bị trùng lặp lại 2 lần (Python, máy ảo Java JVM, ...). Việc này gây tốn tài nguyên của cả máy tính cá nhân, và cũng lãng phí cả RAM, CPU của cụm Kubernetes để chạy 2 pod.

##### Các bước xử lý chính bao gồm:

### (1) Phân tích dữ liệu đầu vào

Các bản ghi JSON được ánh xạ thành lược đồ có cấu trúc với các trường:

- address (district, ward, province, full\_address)
- contact\_info (name, phone)
- title
- description
- estate\_type
- price
- square
- extra\_infos
- post\_date
- post\_id
- link

### (2) Thêm id định danh cho các bản ghi

- Sử dụng trường post\_id để làm id cho bản ghi. Trong trường hợp trường post\_id rỗng, thực hiện băm giá trị link của bản ghi để làm id. Thuật toán băm được sử dụng là MD5. Thêm id vào các bản ghi, điều này để định danh bản ghi trong toàn bộ cơ sở dữ liệu.
- Dữ liệu sau khi đọc từ Kafka và thêm id (bước (1) và (2)) sẽ được ghi vào MinIO dưới dạng file parquet. Lúc này MinIO sẽ lưu một phiên bản dữ liệu không qua xử lý từ các Kafka topic nhưng thêm trường id. Sau đó dữ liệu được xử lý qua các bước bên dưới.

### (3) Xử lý và chuẩn hóa văn bản

Các cột văn bản title, description và estate\_type được xử lý nhằm nâng cao hiệu quả tìm kiếm và phân tích qua quá trình sau:

- Loại bỏ ký tự đặc biệt: Các ký tự Unicode, biểu tượng hoặc ký tự không cần thiết được loại bỏ khỏi nội dung văn bản. Quá trình này đảm bảo văn bản sạch hơn.
- Giới hạn dấu câu lặp:
  - Dấu chấm (.) tối đa 3 lần liên tiếp
  - Các dấu khác (!, ?, ,) tối đa 1 lần
- Chuẩn hóa loại bất động sản:
  - Biệt thự, liền kề → Biệt thự
  - Nhà mặt phố, nhà mặt tiền → Nhà phố
  - Căn hộ, chung cư → Chung cư

Việc chuẩn hóa này giúp đồng nhất dữ liệu và cải thiện độ chính xác khi truy vấn theo từ khóa.

### (4) Xử lý và chuẩn hóa số liệu

Các cột liên quan đến số liệu được xử lý như sau:

- Tính giá trên mét vuông: Tạo cột mới price/square bằng cách chia giá (price) cho diện tích (square).
- Chuẩn hóa giá: Cột price được xử lý để đảm bảo định dạng số hợp lệ:
  - Nếu price đã là số (kiểu int hoặc float), giữ nguyên.
  - Nếu price là chuỗi:

- Thử chuyển sang kiểu float nếu chuỗi là số hợp lệ.
- Nếu chuỗi chứa triệu/m hoặc triệu / m, giá trị được nhân với 106 và diện tích square để tính giá tổng.
- Nếu chuỗi chứa tỷ/m hoặc tỷ / m, giá trị được nhân với 109.
- Nếu không thuộc các trường hợp trên và square là null, trả về null.

### (5) Chuẩn hóa thông tin bổ sung (extra\_infos)

Cột extra\_infos (kiểu map chứa các cặp key-value) được chuẩn hóa thành một cấu trúc có các trường:

- no\_bedrooms: Số phòng ngủ
- no\_bathrooms: Số phòng tắm
- front\_road: Đường trước nhà
- front\_face: Mặt tiền
- no\_floors: Số tầng
- direction: Hướng
- utilization\_square: Diện tích sử dụng
- yo\_construction: Năm xây dựng

Quy trình xử lý:

- Ánh xạ khóa: Các khóa cũ (ví dụ: Số phòng ngủ, Số toilet, Đường trước nhà) được ánh xạ sang các khóa mới tương ứng (như no\_bedrooms, no\_bathrooms, front\_road). Mỗi khóa mới được gán giá trị từ khóa cũ.
- Chuẩn hóa giá trị:
  - Thay dấu phẩy (,) bằng dấu chấm (.) trong các chuỗi số nhằm mục đích ép kiểu về dạng số.
  - Ép kiểu dữ liệu:
    - no\_bedrooms, no\_bathrooms, no\_floors, yo\_construction: Chuyển sang kiểu Integer.
    - front\_road, front\_face, utilization\_square: Chuyển sang kiểu Float, loại bỏ các đơn vị như m, mm, m2.
    - direction: Chuyển sang kiểu String.

### (6) Ghi nhận thời gian xử lý

Một trường created\_at được bổ sung, lưu thời điểm xử lý bản ghi theo định dạng: yyyy/MM/dd HH:mm:ss, biểu thị thời điểm xử lý bản ghi (ví dụ: 2025/05/27 23:11:00).

### (7) Ghi dữ liệu

Dữ liệu sau khi xử lý được ghi vào Elasticsearch để phục vụ truy vấn nhanh và tìm kiếm văn bản, đồng thời sao lưu bản gốc dữ liệu đã xử lý vào MinIO dưới dạng file Parquet để phục vụ batch processing. Việc kết hợp Elasticsearch và MinIO giúp hệ thống vừa tối ưu cho truy vấn thời gian thực, vừa đảm bảo lưu trữ lâu dài với chi phí thấp.

		Account ID: 1920-7672-3134
		LeatherB
(Binh Tân, Đường ... (Bùi Thị Hoàng Y... Nhà mặt phố Tân K...) 99E9  360.0 Bán nhà mặt tiền,... nhamatpho 2025-12-20 16:30:...	Nhà phố (4, null, 12.0, 1... https://bds68.com... 2025/12/20 29572450	4.64999
(Tân Phú, Đường Đ... (Mỹ Trinh, [09081... Nhà mặt tiền kinh... 4E9  42.0 Bán nhà mặt tiền ... nhamatpho 2025-12-20 16:30:...	Nhà phố (2, 2, 20.0, 4.0... https://bds68.com... 2025/12/20 29572482	6
[L, Đường Thới An... [Tí, [0933339xxx] Bán gấp nhà di... 8E9  65.0 Mở bán tiền thuê an ... nhamatpho 2025-12-20 16:31:...	Nhà phố (null, null, 20.0... https://bds68.com... 2025/12/20 28800844	7
[Thị Đức, Đường X... (Long Vũ, [093409... Liên hệ: A. Long(... E10  360.0 Building mt xuâ... nhamatpho 2025-12-20 16:31:...	Nhà phố (2, 2, 20.0, 4.0... https://bds68.com... 2025/12/20 25528860	7.000000
(Phú Nhuận, Đường... (Long Vũ, [093409... Lựa chọn bất động... E10  200.0 Siêu phẩm đầu tư ... nhamatpho 2025-12-20 16:31:...	Nhà phố (50, 50, null, 8... https://bds68.com... 2025/12/20 27174439	3.899999
(Hà Đông, Đường T... (Đỗ Văn Toản, [0... + Mật phô văn mi... E10  67.0 Bán nhà mặt phô t... nhamatpho 2025-12-20 16:32:...	Nhà phố (2, 2, 8.0, 4.5... https://bds68.com... 2025/12/20 29014619	1.799999
(Long Biên, Đường... (Truong Khắc Thủy... Bán đất tặng nhà ... E10  90.0 Bán nhà c4 khu t... nhamatpho 2025-12-20 16:32:...	2.0888888888888898 https://bds68.com... 2025/12/20 29572224	1.880000
(Long Biên, Đường... (Truong Khắc Thủy... Siêu phẩm 90 m² m... E10  90.0 Bán nhà mặt phô p... nhamatpho 2025-12-20 16:32:...	2.944444444444444588 https://bds68.com... 2025/12/20 28835489	2.6499999
[Thanh Xuân, Bán... (Nguyễn Khánh Hải... Bán tòa nhà văn p... E10  65.0 Mở bán phô kinh doan... nhamatpho 2025-12-20 16:32:...	Nhà phố (7, 9, null, 4.0... https://bds68.com... 2025/12/20 29250553	3.500000
(Ba Đình, Đường Đ... (Giang Nguyễn, [0... + Vị trí Vip dung... E10  56.0 Trung tâm bđinh... nhamatpho 2025-12-20 16:32:...	Nhà phố (null, null, null... https://bds68.com... 2025/12/20 29572254	2.520000
(4, Đường Số 4 , ... (Lê Thanh Vũ, [09... +1/ty 300 thương ... E10  80.0 111 ty 300 thương ... nhamatpho 2025-12-20 16:42:...	4.412588 https://bds68.com... 2025/12/20 29572562	1.1299999
(1, Đường Số Vạn ... (Phan Phương Trin... Bán nhà siêu vị t... E10  80.0 Bán nhà siêu vị t... nhamatpho 2025-12-20 16:42:...	Nhà phố (null, null, null... https://bds68.com... 2025/12/20 29572471	3.500000
(Tân Phú, Đường Đ... (Mỹ Trinh, [09081... Nhà mặt tiền kinh... 4E9  42.0 Bán nhà mặt tiền ... nhamatpho 2025-12-20 16:42:...	Nhà phố (2, 2, 20.0, 4.0... https://bds68.com... 2025/12/20 29572482	6
[L, Đường Huỳnh T... (Tí, [0933339xxx] chinh chủ bán gấp... E10  175.0 Nhà mặt tiền huy... nhamatpho 2025-12-20 16:42:...	Nhà phố (4, 5, null, null... https://bds68.com... 2025/12/20 29478021	1.
(4, Đường Số 4 , ... (Tuyên, [0906072x... Nhinh 25 ty - Nhâ... E10  96.0 Nhinh 25 ty nhâ... nhamatpho 2025-12-20 16:42:...	Nhà phố (3, 3, null, 7.0... https://bds68.com... 2025/12/20 29328368	2.5899999
(8, Đường Da Nam ... (Hồ Kim Yên, [090... MẶT TIỀN KDC ĐẠI ... E10  48.0 Mặt tiền kdc dài ... nhamatpho 2025-12-20 16:43:...	1.52380952380952488 https://bds68.com... 2025/12/20 29572529	1.5000000
[L, Đường 14 , Ph... (Chu Đức Tuyên, [... MẶT TIỀN DS145TÂN... E10  68.0 Mặt tiền ds145tân... nhamatpho 2025-12-20 16:43:...	6.85823529411764788 https://bds68.com... 2025/12/20 29572491	2.3000000
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+		
CloudShell Feedback Console Mobile App		
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences		

Hình 11. Log của quá trình xử lý dữ liệu luồng từ Kafka

### b) Xử lý dữ liệu theo lô (Batch Processing)

Bên cạnh xử lý thời gian thực, hệ thống hỗ trợ batch processing để phục vụ các tác vụ phân tích dữ liệu lịch sử.

- Dữ liệu được lưu trữ định kỳ trong MinIO dưới dạng Parquet
- Mỗi file tương ứng với một khoảng thời gian (theo ngày hoặc theo giờ)

#### Triển khai xử lý dữ liệu theo lô:

Tương tự như xử lý luồng, nhóm triển khai xử lý batch bằng cách đóng gói code PySpark bằng Docker và đẩy lên Docker Hub. Tuy nhiên, do công việc xử lý lô dữ liệu đòi hỏi thời gian nên cần chạy định kỳ. Nhóm không trực tiếp triển khai tài nguyên Deployment hay Pod trên cụm Kubernetes tương tự như xử lý luồng, thay vào đó, nhóm sử dụng Airflow để lập lịch tạo pod xử lý lô định kì trên cụm K8s. Sau khi pod hoàn thành nhiệm vụ, Airflow sẽ xóa pod để tiết kiệm tài nguyên cho cụm K8s.

#### Quy trình xử lý batch

Các job batch được thực hiện bằng Spark Batch, bao gồm:

- Tổng hợp số lượng bài đăng theo loại bất động sản
- Phân tích phân bố giá theo khu vực
- Tính toán thống kê trung bình (mean, median, min, max)
- Phát hiện dữ liệu bất thường (outliers)

Dữ liệu sau khi xử lý batch có thể:

- Ghi trả lại Elasticsearch (index batch): Bản ghi được sửa đổi sẽ thay thế bản ghi chưa chính xác trong Elasticsearch, dựa vào trường id để tìm đúng bản ghi.
- Xuất file CSV phục vụ báo cáo

Cung cấp đầu vào cho hệ thống Agent và Frontend

```
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/23 16:33:04 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
25/12/23 16:33:07 WARN MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-sja-file-system.properties,hadoop-metrics2.properties
[1/3] Initializing session
[2/3] Reading data from MinIO...
[3/3] Pushing to ES...
Processing topic: nhamatpho
Text processed.
Numbers processed.
Extra infos processed.
Added created_at field.
Starting to save 128 records to Elasticsearch index: nhapho_index
Successfully saved data to Elasticsearch index: nhapho_index
Processing topic: biethu
Text processed.
Numbers processed.
Extra infos processed.
Added created_at field.
Starting to save 0 records to Elasticsearch index: biethu_index
Successfully saved data to Elasticsearch index: biethu_index
Processing topic: chungcu
Text processed.
Numbers processed.
Extra infos processed.
Added created_at field.
Starting to save 0 records to Elasticsearch index: chungcu_index
Successfully saved data to Elasticsearch index: chungcu_index
Processing topic: nharieng
Text processed.
Numbers processed.
Extra infos processed.
Added created_at field.
Starting to save 0 records to Elasticsearch index: nharieng_index
Successfully saved data to Elasticsearch index: nharieng_index
ubuntu@ip-172-31-26-17:~$
```

Activate Windows  
Go to Settings to activate Windows.

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

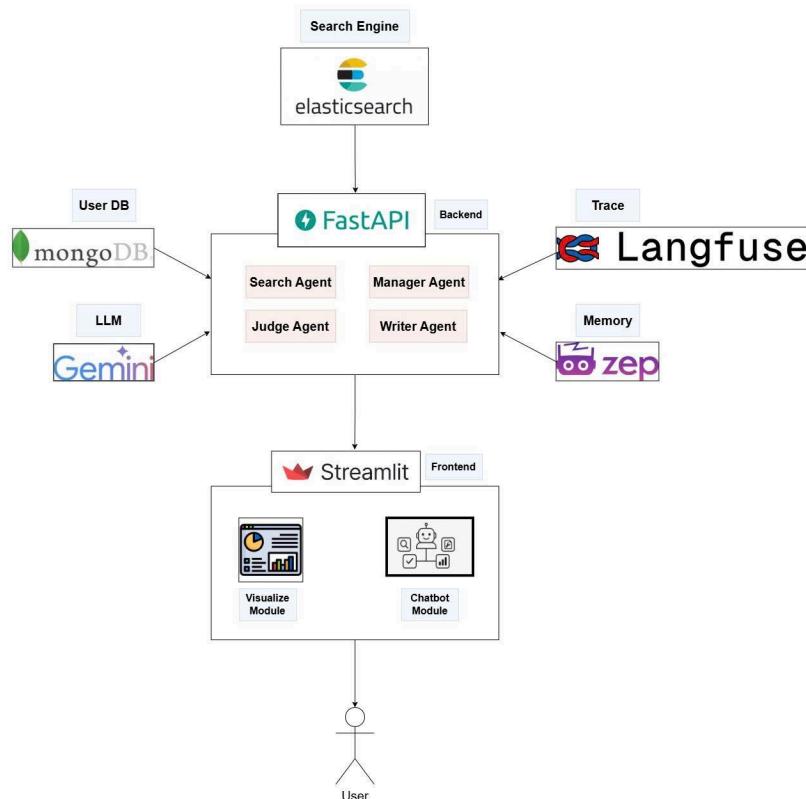
<https://eu-west-2.console.aws.amazon.com/console/home?region=eu-west-2>

**Hình 12.** Log của quá trình xử lý lô dữ liệu

## 5. Triển khai thành phần đa tác tử (Multi-Agent)

Hệ thống Backend được xây dựng trên nền tảng FastAPI, tích hợp các mô hình ngôn ngữ lớn (LLM) để xử lý logic nghiệp vụ phức tạp.

### 5.1. Tổng quan triển khai

**Hình 13.** Triển khai hệ thống đa tác tử

### 5.2. Hệ thống Đa tác tử

Hệ thống bao gồm 5 tác tử chuyên biệt phối hợp theo quy trình tuần tự để xử lý yêu cầu của người dùng:

- **Manager Agent:** Là điểm tiếp nhận truy vấn đầu tiên, chịu trách nhiệm phân tích ý định người dùng bằng kỹ thuật *Few-shot Prompting*. Tác tử này quyết định kích hoạt công cụ `search_db` (nếu hỏi về dữ liệu cụ thể trong DB) hoặc `search_web` (nếu hỏi về thông tin vĩ mô, pháp lý) để định hướng luồng xử lý chính xác.
- **Search Database Agent:** Thực hiện giao tiếp trực tiếp với Elasticsearch để trích xuất dữ liệu. Tác tử sử dụng hai chiến lược truy vấn linh hoạt:
  - *Cơ chế Strict:* Sử dụng logic AND kết hợp Fuzzy Logic cho các trường định lượng (giá, diện tích) để tìm kiếm chính xác nhu cầu.
  - *Cơ chế Flexible:* Sử dụng logic OR để tách nhỏ tiêu chí tìm kiếm, đảm bảo luôn trả về kết quả gợi ý đa dạng ngay cả khi không có bản ghi khớp hoàn toàn.
- **Search Web Agent:** Đóng vai trò bổ sung ngữ cảnh mà cơ sở dữ liệu nội bộ còn thiếu. Thông qua Google Search API, tác tử này thu thập các thông tin thời gian thực như quy hoạch, tiến độ dự án và xu hướng thị trường vĩ mô để làm giàu câu trả lời.
- **Judge Agent:** kiêm định chất lượng đầu ra từ hai tác tử tìm kiếm trên. Chỉ những kết quả có độ phù hợp trên 50% so với câu hỏi gốc mới được chuyển tiếp. Nếu không đạt, tác tử sẽ trả về lý do và yêu cầu bổ sung thông tin, giúp hệ thống có khả năng tự sửa lỗi.
- **Writer Agent:** có vai trò chuyên gia tư vấn để tổng hợp dữ liệu thành báo cáo cuối cùng. Kết quả trả về dưới dạng JSON cấu trúc, bao gồm danh sách bất động sản phù hợp, phân tích chuyên sâu về ưu nhược điểm, và các câu hỏi gợi mở để duy trì hội thoại.

Model sử dụng:

- *Gemini-2.5-Flash:* cho Manager Agent và Search Agent
- *Gemini-2.5-Pro:* cho Judge Agent và Writer Agent

```

⚠ [ZEP] Không lấy được memory (có thể do session mới): status_code: 404, body: message='not found'
tools=['search_db']
['search_db']
User: Tài chính tôi có 10 tỷ, hãy giúp tôi tìm nhà mặt phố ở quận Đồng Da cho gia đình 3 người
👤 Gemini gọi hàm: search_posts_strict
💡 Tham số gốc: {'district': ['Đồng Da'], 'estate_type': ['nhà mặt phố'], 'price': 1000000000.0}
💡 Tham số đã xử lý: {'district': ['Đồng Da'], 'estate_type': ['nhà mặt phố'], 'price': 1000000000.0}
nhamatpho_index
-> DB trả về 3 kết quả.

💡 JUDGE: pass | Reason: I have mapped the user's query for a 'family of 3' to a requirement of at least 2 bedrooms. All 3 retrieved listings (100%) are 'Nhà phố' in 'Đồng Da' district, with prices within the acceptable ±10% range of the user's 10 billion VND budget. Two of the three listings also explicitly meet the 2-bedroom requirement, making the results excellent.
✍ Writer đang viết báo cáo với models/gemini-2.5-pro...
View trace: https://cloud.langfuse.com
Starting research...
 Searching DB (Smart Mode)...
 Judging results (Attempt 1)...
📝 Gemini is analyzing & writing report...

```

**Hình 14.** Tương tác giữa các tác tử trong hệ thống

### 5.3. Các hệ thống hỗ trợ

Để đảm bảo hiệu năng và trải nghiệm người dùng, hệ thống tích hợp các dịch vụ phụ trợ:

- Lưu trữ dữ liệu người dùng (MongoDB):** Quản lý thông tin tài khoản và lịch sử phiên làm việc.

The screenshot shows the MongoDB Atlas interface for a project named 'Project O'. On the left, there's a sidebar with sections for DATABASE (Clusters, Search & Vector Search, Data Explorer, Backup), STREAMING DATA (Stream Processing, Triggers), SERVICES (Migration, Data Federation, Visualization), SECURITY (Quickstart, Project Identity & Access), and a main menu bar at the top. The central area is titled 'Project O Overview' and contains a 'Clusters' section with 'Cluster0' listed, showing a 'Data Size' of 116.32 MB, a 'Browse collections' button, a 'Migrate data' button, and a 'View monitoring' button. Below this is an 'Application Development' section with a 'Connect new' button. To the right is a 'Toolbar' with tabs for 'Resources' and 'Tips (I)', and sections for 'Performance (I)', 'Cost (0)', and 'Resilience (0)'. A green circular button with a checkmark is located at the bottom right of the main area.

Hình 15. Giao diện hệ thống MongoDB

- Truy vết và giám sát (Langfuse):** Ghi lại toàn bộ luồng thực thi của các Agent (Tracing), giúp debug và tối ưu hóa chi phí token cũng như độ trễ của hệ thống.
- Bộ nhớ tác tử (Zep AI):** Hệ thống bộ nhớ chia sẻ giúp các tác tử duy trì ngữ cảnh hội thoại:
  - Bộ nhớ ngắn hạn:* Sử dụng cơ chế cửa sổ trượt (sliding window) lưu 4 lượt tin nhắn gần nhất để xử lý các câu hỏi nối tiếp.
  - Bộ nhớ dài hạn (Knowledge Graph):* Lưu trữ lịch sử dưới dạng đồ thị tri thức (Nodes & Edges). Khi có truy vấn mới, hệ thống sử dụng phương pháp tìm kiếm lai (Hybrid Search: Semantic + Keyword) để trích xuất các sự kiện và thực thể liên quan trong quá khứ, giúp cá nhân hóa câu trả lời.

The screenshot shows the Zep AI platform interface. At the top, it says 'FREE PLAN: Limited to 1,000 Episodes. Upgrade to Flex or Enterprise plans'. The left sidebar has navigation links for Project, Users, Graphs, Playground, Usage & Billing, Members, Documentation, and Feedback. The main area has a 'Demo Project' dropdown and several configuration sections:
 

- Include User Summary in Thread Context:** A toggle switch is turned on.
- Project Keys:** A table showing a key named 'agent\_prj.key' with the value 'z\_1d\*\*\*\*\*gsnw'. There's a '+ Add Key' button.
- Debug Options:** A section for enabling debug logging with a toggle switch.
- API Error Logs:** A table showing logs limited to the last 12 hours or 100 events. Columns include Event, Endpoint, Count, Last Seen, and Detail.

**Hình 16.** Giao diện hệ thống Zep lưu trữ bộ nhớ

#### 5.4. Triển khai API thông qua FastAPI

Dưới đây là chi tiết triển khai các nhóm API chức năng:

##### a) Nhóm API Tương tác với Agents và Quản lý hội thoại

Đây là luồng xử lý chính kết nối người dùng với hệ thống Multi-Agent.

- POST /chat/: Endpoint quan trọng nhất. Khi nhận tin nhắn, FastAPI sẽ kích hoạt Manager Agent. Agent này sẽ truy xuất dữ liệu từ Zep (Memory) và Elasticsearch, sau đó gửi context tới Gemini để sinh câu trả lời. Quá trình này được giám sát bởi Langfuse.
- POST /chat/name\_conversation: Sử dụng LLM để tóm tắt nội dung và tự động đặt tên gợi nhớ cho cuộc hội thoại.

Các API quản lý lịch sử (/chat/save, /chat/history, /chat/{chat\_id}): Đảm bảo trải nghiệm liền mạch, cho phép người dùng xem lại các tư vấn cũ được lưu trữ bền vững trong MongoDB.

##### b) Nhóm API Xác thực người dùng (Authentication)

Hệ thống triển khai cơ chế xác thực bảo mật cơ bản để bảo vệ dữ liệu cá nhân hóa của người dùng:

- POST /register: Xử lý đăng ký với các trường thông tin cơ bản (name, email, password). Mật khẩu được mã hóa (hash) trước khi lưu vào MongoDB.
- POST /login: Cấp phát token truy cập (Session/JWT) khi xác thực thành công, cho phép người dùng truy cập vào các chức năng Chatbot và Dashboard.

##### c) Nhóm API phục vụ Trực quan hóa dữ liệu (Data Visualization APIs)

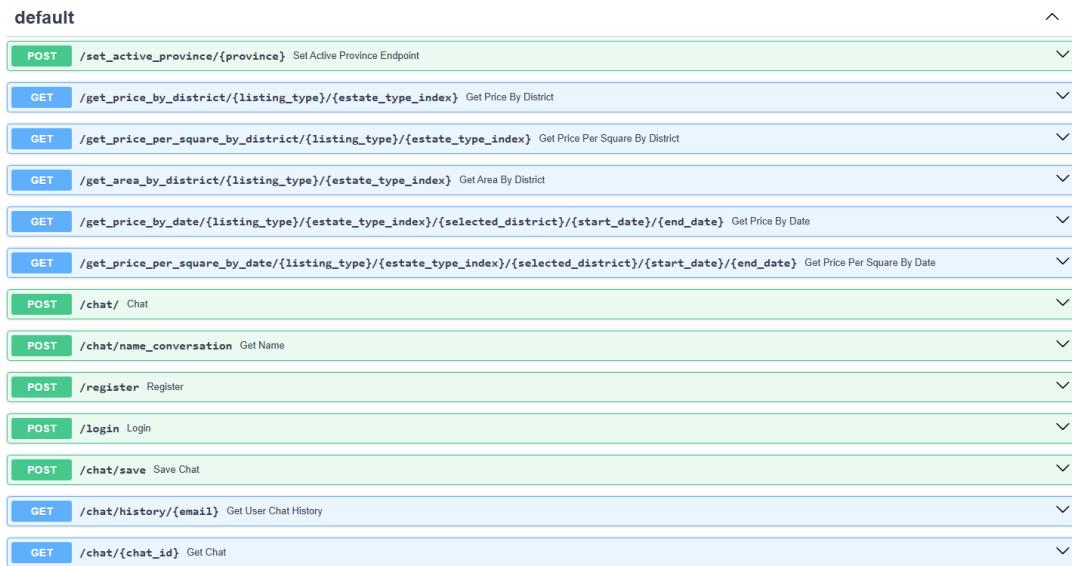
Để hỗ trợ người dùng đưa ra quyết định dựa trên dữ liệu, Backend cung cấp các API phân tích chuyên sâu. Các API này không truy vấn trực tiếp từng bản ghi, mà sử dụng các Aggregation Pipeline của Elasticsearch để tính toán số liệu thống kê từ hàng triệu bản ghi trong thời gian thực.

- Phân tích theo không gian:

- GET /get\_price\_by\_district/{estate\_type}: Trả về giá trung bình theo từng quận/huyện. Dữ liệu này giúp so sánh mức độ đắt đỏ giữa các khu vực.
- GET /get\_area\_by\_district/{estate\_type}: Phân tích diện tích trung bình, giúp người dùng hiểu đặc trưng quy hoạch của từng quận (ví dụ: quận trung tâm diện tích nhỏ, ngoại thành diện tích lớn).
- GET /get\_price\_per\_square\_by\_district/{estate\_type}: Chỉ số đơn giá/m<sup>2</sup> – thước đo chính xác nhất để định giá bất động sản.

- Phân tích theo thời gian:

- GET /get\_price\_by\_date/... và GET /get\_price\_square\_by\_date/...: Trả về chuỗi thời gian (Time-series data) về biến động giá. API này hỗ trợ vẽ các biểu đồ xu hướng, giúp nhận diện chu kỳ tăng/giảm giá của thị trường.



**Hình 17.** Các API trên giao diện FastAPI

## 6. Triển khai thành phần trực quan hóa và giao diện Chatbot

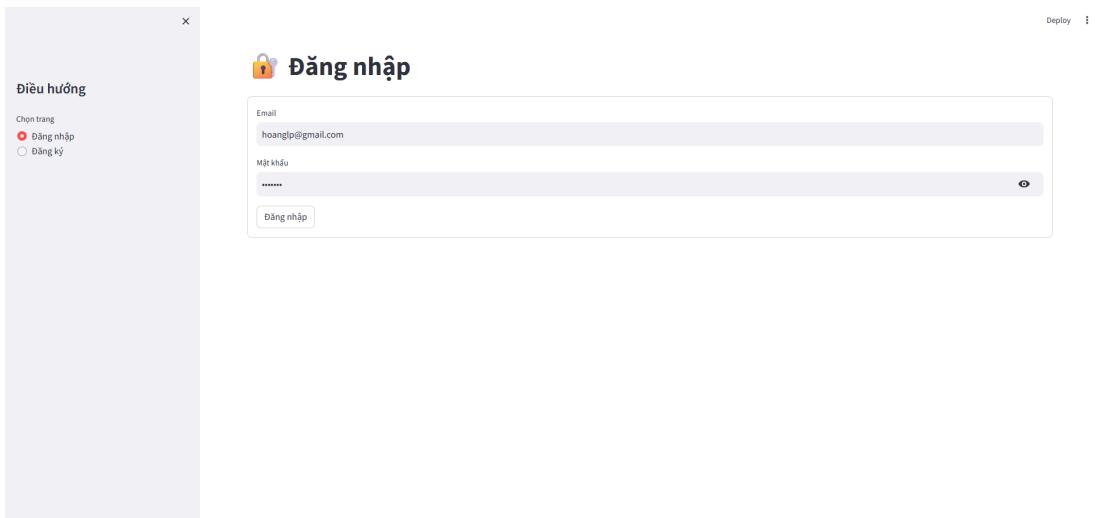
Phân hệ Frontend được phát triển bằng Streamlit, tận dụng nguồn dữ liệu sạch từ Elasticsearch để cung cấp cái nhìn toàn cảnh về thị trường và giao diện tương tác với người dùng.

### 6.1. Giao diện Xác thực người dùng

Giao diện Login/Register được thiết kế tối giản, tập trung vào tính bảo mật và dễ sử dụng. Sau khi đăng nhập thành công, trạng thái phiên (session state) được lưu trữ để duy trì trải nghiệm người dùng.

Sau khi đăng ký, thông tin người dùng sẽ được lưu trong MongoDB. Khi đăng nhập, dữ liệu người dùng nhập vào hệ thống sẽ được dùng để xác thực với cơ sở dữ liệu.

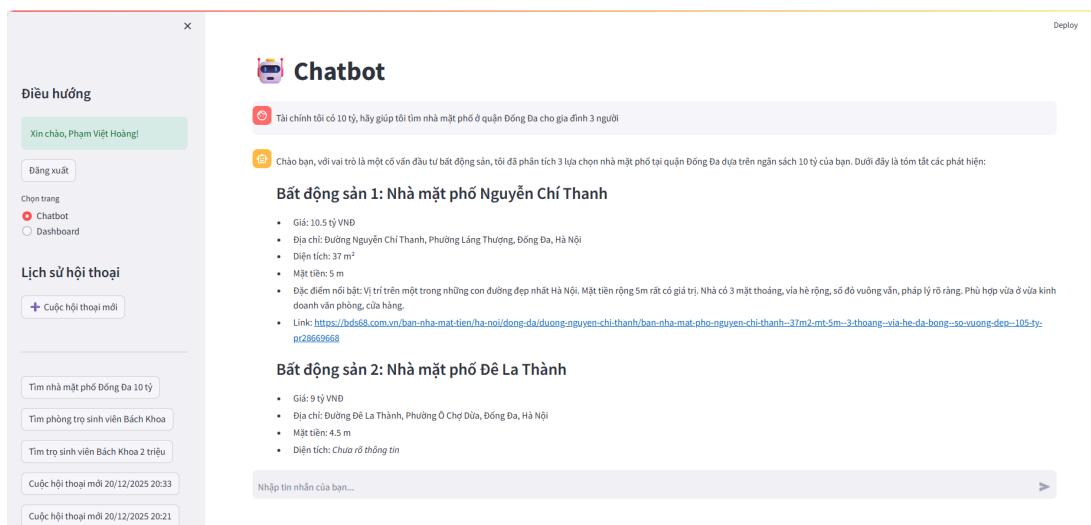
The screenshot shows a Streamlit application window. On the left, there's a sidebar with the title "Điều hướng" (Navigation) and a "Chọn trang" (Select page) section containing two radio buttons: "Đăng nhập" (Login) and "Đăng ký" (Register), with "Đăng ký" selected. The main area is titled "Đăng ký" and contains fields for "Tên của bạn" (Name) with value "Phạm Việt Hoàng", "Email" with value "hoangtp@gmail.com", "Mật khẩu" (Password) with a masked value, and "Xác nhận mật khẩu" (Confirm password) with a masked value. At the bottom is a "Đăng ký" (Register) button.

**Hình 18. Giao diện hệ thống Đăng ký****Hình 19. Giao diện hệ thống Đăng nhập**

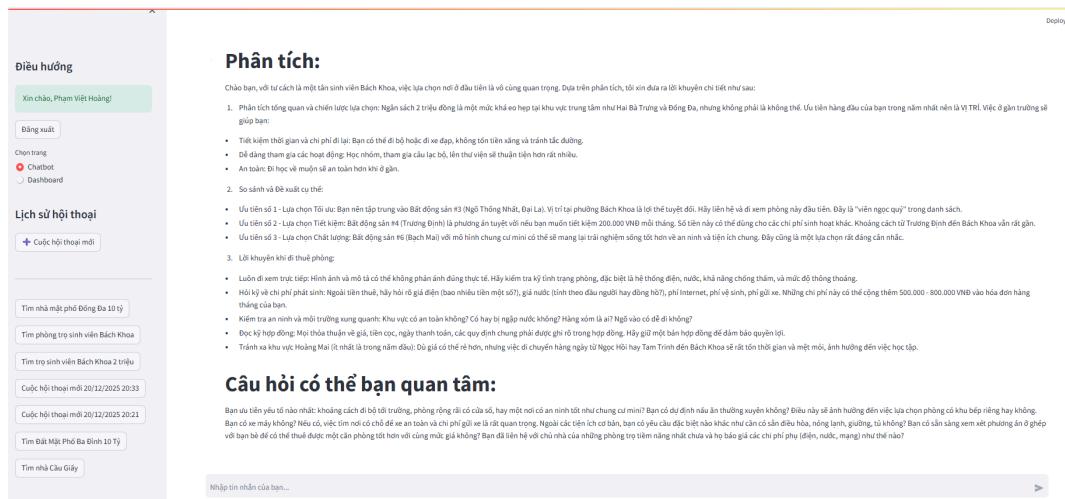
## 6.2. Giao diện Chatbot

Giao diện Chatbot được thiết kế tối giản, đóng vai trò là cầu nối giữa người dùng và hệ thống Backend đa tác tử:

- Tương tác thời gian thực:** Người dùng đặt câu hỏi bằng ngôn ngữ tự nhiên, hệ thống hiển thị trạng thái xử lý của từng Agent (đang tìm kiếm, đang phân tích...) để tăng tính minh bạch.
- Hiển thị kết quả đa phương tiện:** Câu trả lời từ Writer Agent được render dưới dạng Markdown, kết hợp với các thẻ thông tin bất động sản (Card View) có chứa thông số tóm tắt, giúp người dùng dễ dàng tiếp nhận thông tin tư vấn.

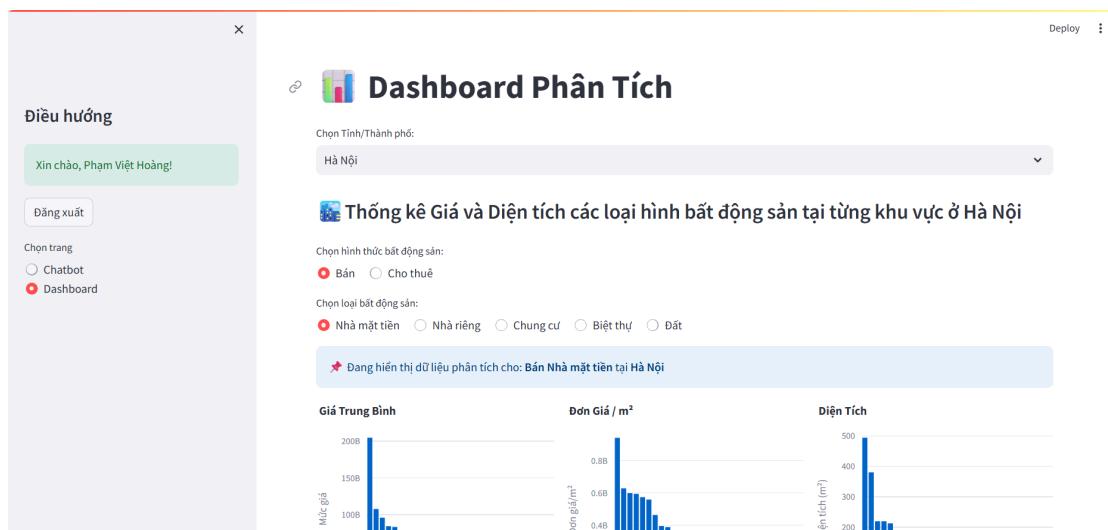


Hình 20. Giao diện hệ thống Chatbot UI



Hình 21. Chatbot phân tích và đưa ra lời khuyên rõ ràng

### 6.3. Thành phần trực quan hóa dữ liệu



Hình 22. Giao diện hệ thống trực quan hóa

Module này biến đổi dữ liệu thô thành các biểu đồ tương tác, hỗ trợ người dùng phân tích thị trường theo ba góc độ:

- So sánh khu vực (Bar Chart):** Các biểu đồ cột so sánh trực quan các chỉ số như Tổng giá trung bình, Đơn giá/m<sup>2</sup> và Diện tích trung bình giữa các quận/huyện ở từng tỉnh thành, theo từng hình thức (mua-bán/cho thuê) và từng loại bất động sản, hỗ trợ việc xếp hạng và lựa chọn khu vực đầu tư phù hợp nhất.



**Hình 23.** Biểu đồ cột thống kê giá và diện tích các loại hình bất động sản tại các quận huyện ở từng tỉnh/thành chọn trước

- Sử dụng dữ liệu từ API /get\_price\_by\_district, /get\_price\_per\_square\_by\_district và /get\_area\_by\_district.
- Mục tiêu: Giúp người dùng nhanh chóng nhận diện:
  - Khu vực nào có giá bất động sản cao nhất/thấp nhất ("Vùng trũng" về giá).
  - Sự phân bố diện tích đặc trưng (ví dụ: tìm khu vực có nhiều căn hộ diện tích lớn cho gia đình).
- Tương tác: Người dùng có thể lọc theo Tỉnh, Hình thức BĐS (mua-bán/cho thuê), loại hình (Chung cư, Nhà đất) để thấy sự khác biệt về giá giữa các phân khúc tại cùng một quận. Đối với bảng, có thể sort theo tăng dần hoặc giảm dần về giá.

STT	Quận/Huyện	Giá TB (VND)	STT	Quận/Huyện	Giá/m <sup>2</sup> (VND)	STT	Quận/Huyện	Diện tích (m <sup>2</sup> )
1	Hoàn Kiếm	204,704,289,000	1	Hoàn Kiếm	941,639,000	1	Sơn Tây	494
2	Ba Đình	107,785,020,000	2	Ba Đình	628,347,000	2	Ba Vì	380
3	Tây Hồ	96,006,269,000	3	Hai Bà Trưng	598,864,000	3	Hoài Đức	220
4	Cầu Giấy	84,458,865,000	4	Tây Hồ	593,769,000	4	Thạch Thất	220
5	Hai Bà Trưng	83,416,233,000	5	Đống Đa	573,067,000	5	Hoàn Kiếm	213
6	Đống Đa	67,620,436,000	6	Cầu Giấy	558,640,000	6	Tây Hồ	172
7	Nam Từ Liêm	59,579,109,000	7	Thanh Xuân	463,105,000	7	Ba Đình	166
8	Thanh Xuân	53,378,072,000	8	Nam Từ Liêm	394,707,000	8	Cầu Giấy	158
9	Bắc Từ Liêm	45,725,440,000	9	Hoàng Mai	387,861,000	9	Mỹ Đức	155
10	Hoàng Mai	44,455,509,000	10	Long Biên	347,404,000	10	Sóc Sơn	153

**Hình 24.** Bảng số liệu thể hiện Giá Trung Bình, Đơn giá/m<sup>2</sup> và Diện Tích của từng các quận huyện ở từng tỉnh thành

- Phân tích xu hướng (Line Chart):** Biểu đồ đường thể hiện biến động giá theo chuỗi thời gian (Time-series). Tính năng này cho phép nhà đầu tư theo dõi lịch

sử tăng/giảm giá của từng loại hình bất động sản để xác định thời điểm mua bán tối ưu.

#### Biểu đồ biến động giá theo thời gian tại từng khu vực ở Đà Nẵng

Chọn quận/huyện để xem lịch sử giá:

Sơn Trà

Từ ngày

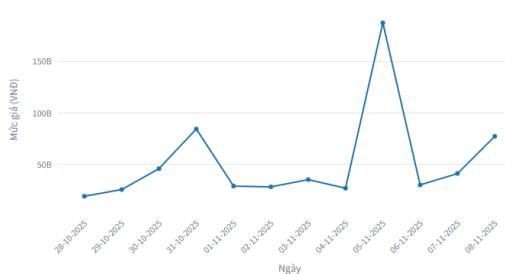
28/10/2025

Đến ngày

08/11/2025

\* Dang hiện thị dữ liệu phân tích cho: **Bán Nhà mặt tiền tại Sơn Trà, Đà Nẵng** từ ngày 28/10/2025 đến ngày 08/11/2025

Tổng Giá Trung Bình



Giá Trung Bình/m<sup>2</sup>



**Hình 25.** Biểu đồ đường thể hiện biến động giá theo từng khoảng thời gian, tại từng khu vực chọn trước

- Sử dụng dữ liệu từ API /get\_price\_by\_date và /get\_price\_per\_square\_by\_date
- Mục tiêu: Giúp người dùng nhanh chóng nhận diện:
  - Sự biến đổi của thị trường qua từng thời điểm tại từng khu vực
  - Giúp người dùng có cái nhìn trực quan, đưa ra được quyết định đầu tư vào thời điểm hợp lý
- Tương tác: Người dùng có thể lọc theo Tỉnh, Hình thức BDS (mua-bán/cho thuê), loại hình (Chung cư, Nhà đất), Thời gian (ngày bắt đầu - ngày kết thúc) để thấy sự khác biệt về giá giữa các phân khúc tại cùng một địa điểm. Đối với bảng, có thể sort theo tăng dần hoặc giảm dần về giá.

Bảng giá theo ngày

Ngày	Giá Trị (VND)
28-10-2025	19,290,909,000
29-10-2025	25,744,444,000
30-10-2025	46,053,529,000
31-10-2025	84,488,966,000
01-11-2025	29,104,000,000
02-11-2025	28,409,091,000
03-11-2025	35,475,833,000
04-11-2025	27,102,353,000
05-11-2025	187,442,857,000
06-11-2025	30,325,000,000

Bảng giá/m<sup>2</sup> theo ngày

Ngày	Giá Trị (VND)
28-10-2025	169,822,000
29-10-2025	235,072,000
30-10-2025	198,199,000
31-10-2025	256,991,000
01-11-2025	191,686,000
02-11-2025	207,884,000
03-11-2025	205,815,000
04-11-2025	185,576,000
05-11-2025	387,646,000
06-11-2025	225,205,000

**Hình 26.** Bảng số liệu thể hiện Giá Trung Bình và Giá/m<sup>2</sup> của từng địa phương trong 1 khoảng thời gian nhất định

- **Phân tích không gian (Heatmap):** Sử dụng thư viện Folium để hiển thị bản đồ nhiệt về giá và mật độ tin đăng. Hệ thống áp dụng thang đo Logarit để cân bằng màu sắc, giúp người dùng dễ dàng nhận diện các vùng trũng về giá hoặc các khu vực đắt đỏ nhất tại từng địa phương.

#### Bản đồ nhiệt giá Nhà mặt tiền theo quận/huyện tại Đà Nẵng

Bản đồ nhiệt giá trung bình



Bản đồ nhiệt giá trung bình/m<sup>2</sup>



**Hình 27.** Bản đồ nhiệt thể hiện mật độ giá trung bình của từng địa phương

- Sử dụng dữ liệu từ API/get\_price\_by\_district, /get\_price\_per\_square\_by\_district
- Mục tiêu: Giúp người dùng nhanh chóng nhận diện:
  - Có cái nhìn tổng quan về khu vực đó, nhìn ra điểm nóng của từng tỉnh/thành. Tính toán, lựa chọn địa điểm đầu tư phù hợp
- Tương tác: Người dùng có thể lọc theo Tỉnh, Hình thức BĐS (mua-bán/cho thuê), loại hình (Chung cư, Nhà đất). Có thể zoom-in hoặc zoom-out với giao diện bản đồ dễ nhìn, quen thuộc, dễ thao tác.

## 7. Source code và documentation

Repository của đề tài được lưu trữ tại GitHub Repository: [real-estate](#). Repository này chứa toàn bộ mã nguồn và tài liệu hướng dẫn để triển khai hệ thống. Toàn bộ các module chính của hệ thống đều được trình bày thành folder code và tài liệu. Ngoài ra, project được mô tả tổng quan trong file README.md, hướng dẫn để triển khai hệ thống được mô tả trong reproduce.md.

## CHƯƠNG 5: TỔNG KẾT VÀ BÀI HỌC KINH NGHIỆM

Dự án đã xây dựng thành công hệ thống phân tích dữ liệu lớn toàn diện cho thị trường bất động sản Việt Nam dựa trên kiến trúc Lambda, đảm bảo khả năng xử lý song song dữ liệu lịch sử và thời gian thực. Bằng việc tích hợp các công nghệ lõi như Apache Kafka, Spark, MinIO và Elasticsearch trên hạ tầng phân tán Kubernetes, hệ thống đã giải quyết hiệu quả bài toán thu thập, làm sạch và chuẩn hóa khối lượng lớn dữ liệu phi cấu trúc từ nhiều nguồn khác nhau.

Giá trị thực tiễn của dự án được thể hiện rõ nét qua phân hệ Frontend Streamlit với khả năng trực quan hóa thị trường sinh động, kết hợp cùng hệ thống Đa tác tử (Multi-Agent) thông minh đóng vai trò như một chuyên gia tư vấn ảo. Kết quả đạt được không chỉ cung cấp một công cụ hỗ trợ ra quyết định đắc lực cho nhà đầu tư mà còn khẳng định khả năng làm chủ công nghệ và tư duy giải quyết vấn đề thực tế của nhóm trong lĩnh vực Big Data.

### 1. Bài học về Thu thập dữ liệu (Data Ingestion)

#### 1.1. Kinh nghiệm về đồng bộ hóa dữ liệu đa nguồn

##### *Problem Description:*

- Bối cảnh: Hệ thống thu thập dữ liệu từ nhiều nguồn (bds.com.vn, bds68...) với cấu trúc HTML, tên trường và định dạng dữ liệu hoàn toàn khác nhau.
- Thách thức: Dữ liệu thô khi đổ về Kafka không đồng nhất (ví dụ: một trang ghi "2 tỷ", trang kia ghi "2000 triệu"), gây khó khăn cho việc định nghĩa Schema thống nhất cho Spark xử lý sau này.
- Tác động hệ thống: Spark Job thường xuyên bị lỗi (Crash) do sai kiểu dữ liệu (Data Type Mismatch) hoặc mất quá nhiều chi phí tính toán để parse dữ liệu tại tầng xử lý.

##### *Approaches Tried:*

- Phương án 1 (Schema-on-Read): Thu thập toàn bộ HTML thô hoặc JSON thô đầy thảng vào Kafka. Việc chuẩn hóa để dành cho Spark làm sau.
  - Nhược điểm: Tăng tải trọng mạng và lưu trữ cho các dữ liệu rác. Logic xử lý của Spark trở nên quá phức tạp và khó bảo trì.
- Phương án 2 (Schema-on-Write): Chuẩn hóa ngay tại tầng Crawler (Scrapy).
  - Ưu điểm: Dữ liệu vào Kafka là dữ liệu sạch, có cấu trúc.
  - Nhược điểm: Tăng độ phức tạp cho Crawler, nếu website nguồn đổi giao diện thì Crawler dễ bị lỗi.

##### *Final Solution:*

- Giải pháp chi tiết: Nhóm quyết định áp dụng Schema-on-Write nhưng ở mức độ linh hoạt. Định nghĩa một Item Class chuẩn (gồm các trường cố định: price, area, location...) ngay trong Scrapy. Mọi dữ liệu crawl về đều được map (ánh xạ) và ép kiểu về format này trước khi serialize thành JSON gửi vào Kafka.
- Kết quả: Dữ liệu trong Kafka và MinIO đạt độ đồng nhất 100% về cấu trúc, giảm thời gian xử lý ETL tại Spark.

##### *Key Takeaways:*

- Technical Insights: Nên xử lý các vấn đề về định dạng càng sớm càng tốt (tại nguồn).

- Best Practices: Sử dụng Pydantic hoặc Avro để validate dữ liệu ngay tại Producer.

## 1.2. Kinh nghiệm về khử trùng lặp nội dung bằng NLP

### **Problem Description:**

- Bối cảnh: Thị trường BDS có đặc thù là một căn nhà được mô giới đăng lại nhiều lần trên cùng một trang hoặc nhiều trang khác nhau (Spam tin đăng).
- Thách thức: Các tin đăng này có tiêu đề khác nhau ("Bán nhà 3 tầng..." vs "Chính chủ bán nhà...") nên không thể lọc bằng URL hay so khớp chuỗi chính xác.
- Tác động hệ thống: Làm loãng kết quả tìm kiếm, lãng phí tài nguyên lưu trữ và làm sai lệch các thống kê giá trung bình.

### **Approaches Tried:**

- Phương án 1: So khớp chính xác trên tiêu đề hoặc mô tả.
  - Nhược điểm: Bỏ sót 90% các tin trùng lặp do chỉ cần thay đổi 1 dấu chấm phẩy là bị coi là tin mới.
- Phương án 2: So khớp theo địa chỉ và giá tiền.
  - Nhược điểm: Không chính xác vì nhiều nhà cùng khu vực có giá giống nhau, hoặc địa chỉ bị viết tắt/ẩn đi.

### **Final Solution:**

- Giải pháp chi tiết: Sử dụng kỹ thuật Content-based Deduplication với NLP.
  1. Tách từ (Tokenize) văn bản tiếng Việt.
  2. Biến đổi văn bản thành Vector tần suất (TF-IDF).
  3. Sử dụng kỹ thuật Băm (LSH - Locality Sensitive Hashing) để gom nhóm các bài viết tương tự nhau vào cùng một "Bucket".
  4. Chỉ so sánh chi tiết trong từng nhóm nhỏ, nếu độ tương đồng Cosine > 50% thì coi là trùng và chỉ giữ lại 1 bản ghi.
- Kết quả: Loại bỏ được tin rác/spam, cải thiện độ sạch của dữ liệu phân tích.

### **Key Takeaways:**

- Technical Insights: Với dữ liệu văn bản phi cấu trúc, các phương pháp so sánh truyền thống là vô nghĩa. Cần dùng Vector Space Model.
- Recommendations: Cần nhắc sử dụng MinHash để tối ưu tốc độ so sánh cho tập dữ liệu lớn hơn.

## 2. Bài học về Xử lý luồng (Stream Processing)

### 2.1. Kinh nghiệm về đảm bảo tính toàn vẹn dữ liệu

### **Problem Description:**

- Bối cảnh: Trong môi trường phân tán, Spark Streaming application có thể bị crash bất cứ lúc nào. Kafka có thể gửi lại tin nhắn (duplicate) hoặc tin nhắn bị mất do mạng (lost).
- Thách thức: Đảm bảo mỗi tin đăng BDS chỉ được lưu vào Database đúng 1 lần duy nhất, không thiếu, không thừa.
- Tác động hệ thống: Nếu xử lý lặp lại -> Dữ liệu thống kê bị nhân đôi sai lệch. Nếu mất tin -> Thiếu dữ liệu thị trường.

### **Approaches Tried:**

- Phương án 1: Checkpoint trên Local Disk.
  - Nhược điểm: Khi Pod Spark bị K8s kill và chuyển sang Node khác, dữ liệu checkpoint cục bộ bị mất -> Spark đọc lại từ đầu (tạo ra hàng loạt bản ghi trùng).
- Phương án 2: Lưu Checkpoint trên MinIO.
  - Ưu điểm: Bền vững, nhưng vẫn có khả năng sinh ra bản ghi trùng nếu Spark crash sau khi ghi DB nhưng trước khi commit offset.

**Final Solution:**

- Giải pháp chi tiết: Kết hợp Checkpointing trên MinIO + Idempotent Sink (Ghi đè thay vì Ghi mới).
  - Nhóm tạo \_id cho mỗi bản ghi Elasticsearch bằng cách băm MD5(URL bài viết).
  - Khi Spark xử lý lại một batch cũ (do crash), nó sẽ ghi đè (Upsert) vào Elasticsearch với cùng \_id. Elasticsearch sẽ tự động cập nhật bản ghi đó thay vì tạo mới.
- Kết quả: Đạt được ngũ nghĩa "Exactly-Once" (chính xác một lần) ngay cả khi hệ thống gặp sự cố.

**Key Takeaways:**

- Technical Insights: Idempotency ở tầng lưu trữ là chốt chặn cuối cùng quan trọng hơn cả logic xử lý ở giữa.
- Best Practices: Luôn thiết kế khóa chính dựa trên nội dung dữ liệu thay vì dùng ID tự tăng.

### 3. Bài học về Lưu trữ dữ liệu

#### 3.1. Kinh nghiệm về tối ưu hóa định dạng lưu trữ

**Problem Description:**

- Bối cảnh: Dữ liệu tích lũy theo thời gian lên tới hàng chục GB.
- Thách thức: Việc đọc lại toàn bộ dữ liệu lịch sử để huấn luyện mô hình hoặc thống kê tồn quá nhiều thời gian I/O và RAM.
- Tác động hệ thống: Các Job Spark Batch chạy rất chậm, thường xuyên bị OOM (Out Of Memory) khi đọc các file JSON/CSV lớn.

**Approaches Tried:**

- Phương án 1: Lưu trữ dạng JSON/CSV (Row-based).
  - Nhược điểm: Dễ đọc bằng mắt thường nhưng cực nặng. Khi cần tính toán cột "Giá", hệ thống vẫn phải load cả cột "Mô tả" rất dài vào RAM.
- Phương án 2: Nén Gzip JSON.
  - Nhược điểm: Tiết kiệm dung lượng đĩa nhưng tốn CPU để giải nén, và vẫn không giải quyết được vấn đề đọc cột thừa.

**Final Solution:**

- Giải pháp chi tiết: Chuyển đổi toàn bộ Data Lake sang định dạng Parquet.
  - Parquet lưu trữ theo cột, cho phép Spark chỉ đọc đúng cột cần thiết
  - Tại tầng Serving (Elasticsearch), sử dụng cơ chế Inverted Index mặc định để tối ưu cho tìm kiếm Full-text.

- Kết quả: Giảm dung lượng lưu trữ trên MinIO và tăng tốc độ truy vấn của Spark Batch.

#### **Key Takeaways:**

- Technical Insights: Định dạng hướng cột là bắt buộc cho các bài toán phân tích (OLAP/Big Data).
- Recommendations: Luôn sử dụng Parquet cho Data Lake.

### **3.2. Kinh nghiệm về phân vùng dữ liệu**

#### **Problem Description:**

- Bối cảnh: Dữ liệu được đẩy vào MinIO liên tục. Sau một tháng, một thư mục có thể chứa hàng vạn file nhỏ.
- Thách thức: Khi muốn truy vấn dữ liệu của "Ngày hôm qua", Spark phải scan toàn bộ danh sách hàng vạn file để lọc, gây nghẽn cổ chai.
- Tác động hệ thống: Thời gian khởi tạo Job lâu, truy vấn chậm.

#### **Approaches Tried:**

- Phương án 1: Lưu tất cả vào một thư mục gốc.
  - Nhược điểm: Không thể quản lý khi số lượng file lớn.
- Phương án 2: Phân vùng theo ID hoặc Hash.
  - Nhược điểm: Không hỗ trợ tốt cho các truy vấn theo khoảng thời gian vốn phô biến trong phân tích thị trường.

#### **Final Solution:**

- Giải pháp chi tiết: Tổ chức thư mục phân cấp theo Hive-style Partitioning: topic=nhamatpho/year=2023/month=10/day=25/data.parquet.
- Implementation: Cấu hình Spark Writer sử dụng partitionBy('topic', 'year', 'month', 'day').
- Kết quả: Spark tận dụng tính năng Partition Pruning, bỏ qua hoàn toàn các thư mục không liên quan khi truy vấn, giúp truy vấn dữ liệu theo ngày gần như tức thì.

#### **Key Takeaways:**

- Best Practices: Thiết kế cấu trúc thư mục cũng chính là thiết kế Index cho File System.
- Recommendations: Chọn khóa phân vùng dựa trên mẫu truy vấn phổ biến nhất (ở đây là Thời gian và Loại BDS).

## **4. Bài học về tích hợp hệ thống**

### **4.1. Kinh nghiệm phát hiện dịch vụ trong môi trường động**

#### **Problem Description:**

- Bối cảnh: Hệ thống triển khai trên AWS EC2. Mỗi khi dừng và khởi động lại Instance (do tiết kiệm chi phí hoặc bảo trì), AWS sẽ cấp phát một địa chỉ Public IP mới.
- Thách thức: Các cấu hình Hard-code IP trong code (Crawler, Airflow connection) liên tục bị sai lệch, làm đứt kết nối giữa máy local và cụm Cloud Cluster.
- Tác động hệ thống: Tốn rất nhiều thời gian cấu hình lại IP thủ công mỗi lần khởi động hệ thống.

**Approaches Tried:**

- Phương án 1: Cấu hình IP tĩnh thủ công.
  - Nhược điểm: Không khả thi trong môi trường Cloud native, gây gián đoạn hệ thống thường xuyên.
- Phương án 2: Sử dụng Dynamic DNS scripts.
  - Nhược điểm: Phức tạp trong cài đặt và có độ trễ khi cập nhật DNS record.

**Final Solution:**

- Giải pháp chi tiết: Áp dụng chiến lược 2 lớp:
  - External Access: Sử dụng AWS Elastic IP để gán 1 địa chỉ IP tĩnh cố định cho Master Node, đảm bảo Crawler từ máy Local luôn tìm thấy đích đến bất kể Instance khởi động lại bao nhiêu lần.
  - Internal Cluster: Tận dụng CoreDNS của Kubernetes. Các dịch vụ gọi nhau thông qua Service Name (ví dụ: kafka-svc, minio-svc) thay vì IP của Pod. K8s sẽ tự động phân giải tên miền nội bộ ra IP mới nhất của Pod.
- Kết quả: Loại bỏ hoàn toàn thời gian chết do đổi IP, hệ thống vận hành liền mạch.

**Key Takeaways:**

- Technical Insights: Trong hệ thống phân tán, đừng bao giờ tin tưởng vào IP động. "Name resolution" (Phân giải tên) là chìa khóa của sự ổn định.
- Best Practices: Luôn sử dụng Service Discovery (K8s Service/DNS) thay vì IP trực tiếp.

## 5. Bài học về Tối ưu hiệu năng

### 5.1. Kinh nghiệm Caching trong tính toán lặp

**Problem Description:**

- Bối cảnh: Quy trình xử lý dữ liệu của Spark thực hiện nhiều bước tuần tự: Đọc -> Làm sạch -> Trích xuất -> Ghi DB. Một số DataFrame trung gian được tái sử dụng nhiều lần.
- Thách thức: Do cơ chế "Lazy Evaluation" của Spark, mỗi khi có một Action (như count, show, write), Spark lại tính toán lại từ đầu chuỗi biến đổi (Lineage) thay vì nhớ kết quả cũ.
- Tác động hệ thống: Thời gian xử lý bị kéo dài gấp 2-3 lần mức cần thiết, lãng phí tài nguyên CPU.

**Approaches Tried:**

- Phương án 1: Không can thiệp (Default Spark behavior).
  - Nhược điểm: Hiệu năng thấp với các thuật toán lặp.
- Phương án 2: Ghi file trung gian ra đĩa (Checkpointing).
  - Nhược điểm: Tốn thời gian I/O đĩa, làm chậm luồng xử lý stream.

**Final Solution:**

- Giải pháp chi tiết: Sử dụng chiến lược In-memory Caching.
  - Gọi hàm .cache() hoặc .persist(StorageLevel.MEMORY\_AND\_DISK) cho các DataFrame quan trọng được truy xuất nhiều lần trong code.

- Spark sẽ lưu kết quả tính toán lần đầu vào RAM (hoặc tràn xuống Disk nếu thiếu RAM), các bước sau chỉ cần đọc từ Cache.
- Kết quả: Giảm thời gian thực thi các Job phức tạp xuống đáng kể.

**Key Takeaways:**

- Recommendations: Chỉ Cache những DataFrame thực sự được tái sử dụng. Cache bừa bãi sẽ gây lỗi OOM.

**6. Bài học về mở rộng hệ thống****6.1. Kinh nghiệm về Hybrid Scaling****Problem Description:**

- Bối cảnh: Ban đầu nhóm triển khai trên các Instance nhỏ (t3.micro/small) để tiết kiệm chi phí.
- Thách thức: Khi triển khai đồng thời Kafka, Spark và Elasticsearch, các máy ảo thường xuyên bị treo cứng, SSH không phản hồi do tràn RAM và CPU quá tải.
- Tác động hệ thống: Không thể nhập lệnh, Cluster sập hoàn toàn, mất dữ liệu đang xử lý.

**Approaches Tried:**

- Phương án 1 (Scale Out - Ngang): Thêm nhiều node cấu hình yếu.
  - Nhược điểm: Quản lý quá nhiều node phức tạp, chi phí mạng giữa các node tăng, trong khi Java (JVM) của các dịch vụ Big Data như ES/Kafka rất "đói" RAM, máy yếu không chạy nổi dù chỉ 1 Pod.
- Phương án 2 (Scale Up - Dọc): Dùng 1 máy cực mạnh.
  - Nhược điểm: Điểm chết duy nhất (Single Point of Failure), chi phí đắt đỏ.

**Final Solution:**

- Giải pháp chi tiết: Kết hợp Scale Up và Scale Out hợp lý.
  - Hạ tầng: Nâng cấp cấu hình Instance lên loại m7i-flex (dòng tối ưu cho Memory/Compute cân bằng) để đảm bảo mỗi Node đủ sức "gánh" các Java Application nặng.
  - Ứng dụng: Tăng số lượng Pod (Kafka Broker, Spark Worker) để phân tải xử lý song song.
- Kết quả: Hệ thống ổn định trên 4 Instance m7i-flex, loại bỏ hoàn toàn hiện tượng lag/treo máy.

**Key Takeaways:**

- Technical Insights: Với các ứng dụng Big Data chạy trên JVM (Java Virtual Machine), cấu hình RAM của từng Node quan trọng hơn số lượng Node.

**7. Bài học về bảo mật và quản trị hệ thống****7.1. Kinh nghiệm về quản lý truy cập****Problem Description:**

- Bối cảnh: Airflow chạy trên máy cá nhân (Local) cần quyền điều khiển cụm K8s trên Cloud (Remote). Đồng thời, Crawler cần đẩy dữ liệu vào Kafka nằm sâu trong mạng riêng ảo (VPC).

- Thách thức: Mặc định AWS chặn mọi kết nối lạ. Kubernetes mặc định không cho phép người ngoài can thiệp vào Pod.
- Tác động hệ thống: Kết nối bị từ chối, không thể deploy job hoặc đẩy dữ liệu.

#### *Approaches Tried:*

- Phương án 1: Thiết lập VPN Site-to-Site hoặc Bastion Host.
  - Nhược điểm: Quá phức tạp và tốn kém thời gian cho quy mô bài tập lớn.
- Phương án 2: Mở cửa hoàn toàn.
  - Ưu điểm: Nhanh, dễ debug.

#### *Final Solution:*

- Giải pháp chi tiết:
  - Network: Cấu hình Security Group (Tường lửa) cho phép All TCP Traffic (0.0.0.0/0) để đảm bảo thông suốt mọi kết nối trong quá trình phát triển (chấp nhận rủi ro trong môi trường Lab).
  - K8s RBAC: Tạo ServiceAccount và ClusterRoleBinding cấp quyền admin giới hạn cho Airflow để nó có thể tạo/xóa Pod Spark Batch mà không cần can thiệp thủ công.
- Kết quả: Airflow điều phối trơn tru, Crawler đẩy dữ liệu thành công.

#### *Key Takeaways:*

- Best Practices: Trong môi trường Production, tuyệt đối không mở All TCP. Nên dùng VPN hoặc Whitelist IP cụ thể. Tuy nhiên, với dự án học thuật, sự đơn giản được ưu tiên để tập trung vào logic xử lý dữ liệu.

## 8. Bài học về khả năng chịu lỗi

### 8.1. Kinh nghiệm thiết kế hệ thống với tư duy "Crash-first"

#### *Problem Description:*

- Bối cảnh: Trong hệ thống phân tán, lỗi phần cứng hoặc phần mềm là điều chắc chắn sẽ xảy ra. Pod có thể bị K8s kill để giải phóng tài nguyên bất cứ lúc nào.
- Thách thức: Đảm bảo hệ thống tự phục hồi và không mất dữ liệu khi có sự cố.

#### *Approaches Tried:*

- Phương án 1: Không có cơ chế dự phòng (Replication Factor = 1).
  - Nhược điểm: Mất 1 Node là mất dữ liệu vĩnh viễn. Hệ thống dừng hoạt động.

#### *Final Solution:*

- Giải pháp chi tiết:
  - Data Replication: Cấu hình Kafka replication-factor = 2 nhưng số lượng nhân bản của Elasticsearch vẫn là 1 để đảm bảo tài nguyên. Nếu 1 Broker chết, Broker khác lập tức thay thế, dữ liệu vẫn an toàn.
  - Disaster Recovery: Coi MinIO (Data Lake) chứa dữ liệu thô là sự thật. Nếu Elasticsearch bị lỗi index hoặc crash hỏng dữ liệu, nhóm có quy trình chạy lại Spark Batch để Re-index toàn bộ dữ liệu từ MinIO sang Elasticsearch.
- Kết quả: Hệ thống đạt độ sẵn sàng cao.

#### *Key Takeaways:*

- Technical Insights: Storage rẻ hơn Data. Đừng tiết kiệm dung lượng lưu trữ mà hy sinh khả năng an toàn dữ liệu. Luôn giữ lại Raw Data.

## 9. Bài học về hệ thống Backend & Multi-agent

### 9.1. Kinh nghiệm về quản lý ngữ cảnh và tối ưu chi phí Token

#### *Problem Description:*

- Bối cảnh: Chatbot cần duy trì hội thoại dài để hiểu sở thích người dùng. Ban đầu, nhóm gửi toàn bộ lịch sử chat vào prompt mỗi lần gọi API.
- Thách thức: Khi hội thoại kéo dài quá 10 câu, số lượng token đầu vào tăng vọt, nhanh chóng chạm giới hạn của Gemini Flash.
- Tác động hệ thống: Chi phí API tăng phi mã theo cấp số cộng. Tốc độ phản hồi chậm đi đáng kể do mô hình phải xử lý lượng văn bản đầu vào quá lớn.

#### *Approaches Tried:*

- Phương án 1: Gửi toàn bộ lịch sử.
  - Nhược điểm: Tốn kém và dễ gây lỗi Token Limit Exceeded.
- Phương án 2: Tóm tắt hội thoại. Sử dụng một LLM call phụ để tóm tắt các đoạn chat cũ.
  - Nhược điểm: Tăng độ trễ vì phải chờ tóm tắt xong mới trả lời được. Có rủi ro mất mát các chi tiết nhỏ (ví dụ: con số cụ thể về giá tiền).

#### *Final Solution:*

- Giải pháp chi tiết: Chuyển sang kiến trúc Bộ nhớ phân tầng (Hybrid Memory).
  - Ngắn hạn: Sử dụng cơ chế cửa sổ trượt (Sliding Window), chỉ giữ nguyên văn 4 tin nhắn gần nhất để duy trì mạch chuyện tức thời.
  - Dài hạn: Tích hợp Zep AI để lưu trữ dưới dạng Graph. Thay vì gửi văn bản, hệ thống trích xuất các "Facts" (Sự kiện/Sở thích) và lưu vào Vector Store. Khi cần, chỉ truy xuất các Fact liên quan.
- Kết quả: Giảm lượng token tiêu thụ cho các hội thoại dài, duy trì độ trễ ổn định dưới mức tối thiểu.

#### *Key Takeaways:*

- Technical Insights: "Less is More". Việc chọn lọc thông tin (Information Retrieval) quan trọng hơn là nhồi nhét dữ liệu vào Context Window.
- Recommendations: Luôn tách biệt Memory thành Short-term và Long-term cho các ứng dụng Chatbot phức tạp.

### 9.2. Kinh nghiệm về kiểm soát ảo giác của Agent

#### *Problem Description:*

- Bối cảnh: Hệ thống sử dụng Search Agent để tìm dữ liệu BDS, sau đó chuyển cho Writer Agent để viết lời khuyên.
- Thách thức: Fuzzy Search đôi khi trả về kết quả không chính xác (ví dụ: tìm "nhà quận 1" ra "nhà quận 12"). Writer Agent tin tưởng tuyệt đối vào dữ liệu này và viết ra các lời khuyên sai sự thật.
- Tác động hệ thống: Người dùng mất niềm tin vào độ chính xác của hệ thống tư vấn.

#### *Approaches Tried:*

- Phương án 1: Tuyên tính đơn giản (Search -> Write).
  - Nhược điểm: Không có cơ chế kiểm lỗi. Sai từ đầu vào dẫn đến sai đầu ra (Garbage In, Garbage Out).
- Phương án 2: Cải thiện Prompt cho Search Agent.
  - Nhược điểm: Dù prompt tốt đến đâu, các search engine (Elasticsearch) vẫn có tỷ lệ sai số nhất định mà LLM không kiểm soát được.

**Final Solution:**

- Giải pháp chi tiết: Triển khai mô hình Human-in-the-loop giả lập
  - Thêm một bước trung gian: Judge Agent. Tác tử này đóng vai trò "Biên tập viên", nhận đầu vào là Yêu cầu người dùng và Kết quả tìm kiếm.
  - Nó so sánh và chấm điểm độ liên quan (Relevance Score). Nếu điểm thấp (<50%), nó từ chối và yêu cầu Search Agent tìm lại với từ khóa khác (Self-correction).
- Kết quả: Loại bỏ hầu như toàn bộ các câu trả lời sai lệch về địa điểm và giá cả.

**Key Takeaways:**

- Best Practices: "Trust but Verify". Không bao giờ nói thẳng đầu ra của công cụ tìm kiếm vào đầu vào của bộ sinh văn bản mà không qua kiểm chứng.

**9.3. Kinh nghiệm Debugging trong hệ thống bất định****Problem Description:**

- Bối cảnh: Hệ thống có 5 Agent gọi nhau liên tục. Logic của LLM là bất định (Non-deterministic), cùng một input có thể ra output khác nhau.
- Thách thức: Khi câu trả lời bị sai, nhóm không biết lỗi nằm ở đâu: Do Manager định tuyến sai? Do Search tìm không ra? Hay do Writer viết kém?
- Tác động hệ thống: Mất hàng giờ đồng hồ để dò lỗi thủ công qua các dòng print() trong console log hỗn độn.

**Approaches Tried:**

- Phương án 1: Sử dụng print() statement.
  - Nhược điểm: Không thể theo dõi được luồng dữ liệu qua nhiều bước phức tạp.
- Phương án 2: Ghi log vào file text.
  - Nhược điểm: Khó hình dung được độ trễ của từng bước và chi phí token.

**Final Solution:**

- Giải pháp chi tiết: Tích hợp Langfuse cho Tracing (Truy vết).
  - Langfuse tự động vẽ lại biểu đồ luồng chạy.
  - Nhóm có thể bấm vào từng node để xem chính xác: Prompt gửi đi là gì? Model trả về gì? Tốn bao nhiêu giây?
- Kết quả: Giảm thời gian debug trung bình từ 2 giờ xuống còn 15 phút. Dễ dàng phát hiện các điểm bottleneck về hiệu năng.

**Key Takeaways:**

- Technical Insights: Với ứng dụng LLM, Observability quan trọng ngang với Code Logic.

## 10. Bài học về Trực quan hóa dữ liệu

### 10.1. Kinh nghiệm về xử lý phân phối dữ liệu lệch trong Heatmap

#### *Problem Description:*

- Bối cảnh: Dữ liệu giá bất động sản thường tuân theo quy luật phân phối Long-tail distribution. Đa số các căn nhà có giá từ 3-10 tỷ đồng, nhưng tồn tại một số ít bất động sản "siêu sang" (biệt thự, nhà phố cổ) có giá lên tới hàng trăm, thậm chí hàng nghìn tỷ đồng.
- Thách thức: Khi vẽ Bản đồ nhiệt (Heatmap) về giá, nếu sử dụng thang đo tuyến tính mặc định, dải màu sẽ bị chi phối hoàn toàn bởi các giá trị ngoại lai (outliers) khổng lồ này.
- Tác động hệ thống: Bản đồ bị hiện tượng "Washed out" (Bạc màu). Chỉ có 1-2 điểm tại trung tâm thành phố hiện màu Đỏ (giá cao nhất), trong khi 99% các khu vực còn lại đều hiển thị màu Xanh (giá thấp nhất). Người dùng không thể phân biệt được đâu là khu vực "bình dân" ( $30\text{tr}/\text{m}^2$ ) và "trung cấp" ( $60\text{tr}/\text{m}^2$ ) vì tất cả đều bị coi là "rất thấp" so với khu vực  $1\text{tỷ}/\text{m}^2$ .

#### *Approaches Tried:*

- Phương án 1: Sử dụng thang đo tuyến tính (Min-Max Scaling).
  - Nhuộc điểm: Như đã nêu trên, độ tương phản cực kém. Bản đồ gần như vô dụng để so sánh các khu vực ngoại thành.
- Phương án 2: Loại bỏ giá trị ngoại lai (Trimming/Clipping). Cắt bỏ top 5% giá cao nhất.
  - Nhuộc điểm: Làm mất đi thông tin về phân khúc thị trường cao cấp – vốn là dữ liệu quan trọng mà nhà đầu tư quan tâm. Dữ liệu hiển thị không còn trung thực.

#### *Final Solution:*

- Giải pháp chi tiết: Áp dụng Thang đo Logarit cho dải màu.
  - Thay vì ánh xạ trực tiếp Giá tiền  $\rightarrow$  Màu sắc, hệ thống ánh xạ  $\text{Log}(\text{Giá tiền}) \rightarrow$  Màu sắc.
  - Phép biến đổi Logarit giúp "nén" khoảng cách giữa các giá trị lớn, làm cho sự chênh lệch giữa 5 tỷ và 10 tỷ trở nên rõ ràng hơn, tương đương với sự chênh lệch giữa 50 tỷ và 100 tỷ trên thang màu.
- Implementation: Sử dụng hàm `numpy.log1p()` trong Python để tiền xử lý dữ liệu trước khi đưa vào thư viện Folium.
- Kết quả: Bản đồ nhiệt hiển thị dải màu hài hòa. Người dùng có thể nhìn thấy rõ các "vùng nóng" cục bộ tại các quận xa trung tâm, giúp việc đánh giá tiềm năng tăng trưởng chính xác hơn.

#### *Key Takeaways:*

- Technical Insights: Với dữ liệu tài chính hoặc bất động sản, thang đo tuyến tính hiếm khi là lựa chọn đúng. Luôn kiểm tra biểu đồ phân phối (Histogram) của dữ liệu trước khi chọn thang màu.
- Best Practices: Sử dụng Log Scale hoặc Quantile Scale cho Heatmap để đảm bảo độ phân giải hình ảnh tốt nhất cho mọi khoảng giá trị.

## **TÀI LIỆU THAM KHẢO**

- [1] TS. Trần Việt Trung, *Slide bài giảng: Lưu trữ và xử lý dữ liệu lớn*. Trường Công nghệ Thông tin & Truyền thông - Đại học Bách Khoa Hà Nội, 2024.
- [2] Apache Software Foundation, "Apache Kafka Documentation," *Apache Kafka*.
- [3] Apache Software Foundation, "Apache Spark Documentation: Structured Streaming Programming Guide," *Apache Spark*.
- [4] Apache Software Foundation, "Apache Airflow Documentation," *Apache Airflow*.[6] Docker Inc., "Docker Documentation," *Docker*.
- [5] The Kubernetes Authors, "Kubernetes Documentation,"
- [6] Scrapy Community, "Scrapy 2.11 Documentation," *Scrapy*.
- [7] Elastic, "Elasticsearch Guide," *Elastic*.
- [8] MongoDB, "MongoDB Manual," *MongoDB Documentation*.
- [9] Google AI, "Gemini API Overview & Python SDK," *Google AI for Developers*.
- [10] Zep AI, "Zep: Long-term Memory for AI Assistants," *Zep Documentation*.
- [11] Langfuse, "Langfuse Documentation: Open Source LLM Engineering Platform," *Langfuse*.
- [12] S. Ramirez, "FastAPI Documentation," *FastAPI*.
- [13] Streamlit Inc., "Streamlit Documentation," *Streamlit*.